Visual Studio 2005 Visual Basic

Copyright© 2016 Microsoft Corporation

Le contenu de ce document est obsolète et n'est plus mis à jour ni pris en charge. Certains liens peuvent ne pas fonctionner. Le contenu obsolète représente la dernière version mise à jour de ce contenu.

Visual Basic

Microsoft Visual Basic 2005 est une évolution du langage Visual Basic qui est conçue pour générer des applications de type sécurisé et orientées objet de manière productive. Visual Basic permet aux développeurs de cibler des périphériques Windows, Web et mobiles. Comme avec tous les langages ciblant le Microsoft .NET Framework, les programmes écrits en Visual Basic bénéficient de la sécurité et de l'interopérabilité entre les langages.

Cette génération de Visual Basic perpétue la tradition et propose un moyen rapide et aisé de créer des applications basées sur le .NET Framework.

Cette version de Visual Basic prend à nouveau en charge l'opération Modifier & Continuer et propose de nouvelles fonctionnalités pour le développement rapide d'applications. L'une de ces fonctionnalités, appelée **My**, donne un accès rapide aux tâches courantes proposées par le .NET Framework, ainsi qu'aux informations et instances d'objet par défaut liées à l'application et à son environnement d'exécution. Les nouvelles fonctionnalités de langage comptent notamment la continuation de boucle, la mise à disposition de ressources garanties, la surcharge d'opérateur, les types génériques et les événements personnalisés. Visual Basic intègre également l'intégralité du .NET Framework et du Common Language Runtime, qui offrent ensemble l'interopérabilité entre les langages, le garbage collection, une sécurité améliorée et une meilleure prise en charge du versioning.

Dans cette section

Mise en route de Visual Basic

Vous aide à commencer à travailler en répertoriant les éléments nouveaux et disponibles dans plusieurs éditions du produit.

Visite guidée de Visual Basic

Décrit étape par étape les différents aspects de la programmation avec Visual Basic.

Aide pour les utilisateurs de Visual Basic 6.0

Décrit les différences dans Visual Basic entre la version 6.0 et la version actuelle.

Mise à niveau d'applications dans Visual Basic

Montre comment mettre à niveau différents types de code Visual Basic.

Développement d'applications avec Visual Basic

Décrit les différents aspects du développement dans Visual Basic, tels que l'éditeur de code, la sécurité, la gestion des exceptions, le débogage et l'utilisation de la bibliothèque de classes .NET Framework.

Guide de programmation Visual Basic

Présente les composants essentiels de Visual Basic, langage de programmation orienté objet.

Référence (Visual Basic)

Contient les informations sur le langage et le compilateur Visual Basic.

Exemples d'applications Visual Basic

Contient des informations sur les exemples.

Rubriques connexes

Introduction à Visual Studio

Offre des informations sur les nouvelles fonctionnalités, une vue d'ensemble détaillée de Visual Studio et du .NET Framework, ainsi que des conseils de mise en route.

Environnement de développement intégré pour Visual Studio

Décrit les outils partagés qui permettent de concevoir, développer, déboguer et déployer des applications.

Applications, composants et services Windows

Décrit les applications Windows, les composants de différents types, les services Web XML, les services Windows, les applications console et les applications 64 bits.

Programmation du .NET Framework dans Visual Studio

Décrit le développement d'applications avec le .NET Framework.

Visual Web Developer

Décrit comment créer des applications Web.

Référence de la bibliothèque de classes .NET Framework

Fournit une entrée dans la bibliothèque de classes, interfaces et types de valeurs inclus dans le Kit de développement .NET Framework SDK de Microsoft.

Visual Studio Tools pour Office

Vous montre comment tirer profit des améliorations de productivité de Microsoft .NET Framework version 2.0 pour étendre Microsoft Office Word 2003, Microsoft Office Excel 2003 et Microsoft Office Outlook 2003.

Développement Smart Device

Décrit le développement de logiciels qui s'exécutent sur des smart devices Windows CE, tels que des Pocket PC et des Smartphone.

Mise en route de Visual Basic

Cette section de la documentation vous permet de mettre en route le développement d'applications Visual Basic.

Dans cette section

Nouveautés de Visual Basic

Aborde les nouvelles fonctionnalités de Visual Basic.

Éditions Visual Basic

Décrit les fonctionnalités qui accompagnent Visual Studio et le .NET Framework.

Procédures pas à pas relatives au langage Visual Basic

Fournit une liste des pages d'aide qui illustrent les aspects du langage Visual Basic.

Ressources supplémentaires pour les programmeurs Visual Basic

Fournit une liste de sites Web et de newsgroups susceptibles de vous aider à trouver des solutions à des problèmes courants.

Comment faire en Visual Basic

Fournit des pages par catégorie de liens vers les pages d'aide « Comment » sur les sujets de programmation importants.

Rubriques connexes

Choix des technologies et outils à utiliser

Fournit un guide interactif pour vous aider à choisir la meilleure approche ou technologie à utiliser, ainsi que des liens vers des informations supplémentaires.

Programmation orientée objet dans Visual Basic

Fournit des liens vers les pages qui présentent la programmation orientée objet et décrivent comment créer vos propres objets et comment utiliser des objets pour simplifier votre codage.

Exemples d'applications Visual Basic

Fournit des liens vers des exemples de code en Visual Basic.

Utilisation de l'aide de Visual Studio

Explique comment utiliser des filtres et fournit des liens vers des pages sur la recherche d'informations dans le système d'aide, sur l'utilisation de l'index et sur la recherche d'informations dans MSDN Library.

Personnalisation de l'environnement de développement

Explique la procédure à suivre pour définir les options dans l'environnement de développement intégré (IDE, Integrated Development Environment), notamment les touches de raccourci et les préférences de débogage.

Support technique

Décrit la procédure à suivre pour recevoir de l'aide de la part des services de Support Technique et envoyer des commentaires.

Documentation Visual Studio

Fournit des liens vers des rubriques de la documentation Visual Studio.

Éditions de Visual Studio

Décrit les fonctionnalités des trois éditions de Visual Studio.

Visual C#

Fournit des liens vers des rubriques de la documentation sur le développement d'applications à l'aide de Visual C#.

Visual C++

Fournit des liens vers la documentation Visual C++.

Visual J#

Fournit des liens vers la documentation sur le développement d'applications à l'aide de Visual J#.

Visual Studio Tools pour Office

Fournit des informations sur l'utilisation de Microsoft Office et de Visual Studio en tant que partie intégrante de votre application d'entreprise.

Nouveautés de Visual Basic

Cette page répertorie les fonctionnalités nouvelles et améliorées disponibles dans cette version de Visual Basic dans Visual Studio 2005. Faites votre choix parmi les liens ci-dessous pour obtenir plus de détails sur chacune de ces fonctionnalités.

Remarque

Si vous passez de Visual Basic 6.0 à Visual Basic 2005, plusieurs rubriques vous aideront à effectuer la mise à niveau vers la nouvelle version. Pour plus d'informations, consultez Nouveautés pour les utilisateurs de Visual Basic 6.0.

Compilateur et langage Visual Basic

Les améliorations apportées au langage dans Visual Basic 2005 simplifient le code source et autorisent une interaction avec des composants qui utilisent des fonctionnalités avancées. Pour plus d'informations, consultez Nouveautés du langage Visual Basic et Nouveautés du compilateur Visual Basic.

Développement Visual Basic avec My

Visual Basic 2005 fournit de nouvelles fonctionnalités destinées au développement rapide d'application. Celles-ci ont pour objectif d'améliorer la productivité et de faciliter l'exploitation, tout en fournissant de la puissance. L'une de ces fonctionnalités, nommée **My**, fournit un accès aux fonctionnalités les plus couramment utilisées du .NET Framework, ainsi que des informations et des instances d'objet par défaut associées à l'application et à son environnement d'exécution. Ces informations sont organisées dans un format qui autorise une navigation IntelliSense et dont la logique s'organise en fonction de l'utilisation. Pour plus d'informations, consultez Développement avec My.

Développement d'applications

L'My.Application, objet donne accès à une application, en vous permettant de la mettre à jour facilement et de consigner des informations à son sujet. **My.Application** peut également être utilisé pour accéder aux arguments de ligne de commande d'une application ou exécuter du code lorsque l'application démarre ou se termine. Pour plus d'informations, consultez Mise à jour d'une application, Enregistrement d'informations provenant de l'application et Accès à une application en cours d'exécution dans Visual Basic.

Presse-papiers

L'My.Computer.Clipboard, objet facilite l'accès au Presse-papiers en lecture et en écriture, et fournit des méthodes pour en effacer le contenu et déterminer le type des données qu'il stocke. Pour plus d'informations, consultez Stockage de données dans le Presse-papiers et lecture du Presse-papiers.

Ressources d'ordinateur

Beaucoup de ressources de l'ordinateur sont accessibles de la même manière par le biais d'objets **My**, notamment l'horloge, le clavier, la souris, les ports et le Registre de l'ordinateur. Pour plus d'informations, consultez Accès aux ressources de l'ordinateur.

Entrées/sorties de fichier

L'My.Computer.FileSystem, objet fournit des méthodes et des propriétés d'accès aux fichiers qui simplifient les E/S de fichier. L'TextFieldParser, objet vous permet d'analyser des fichiers de texte volumineux délimités ou de largeur fixe, tandis que des méthodes telles que **WriteAllText**, **WriteAllBytes**, **ReadAllText** et **ReadAllBytes** rendent l'accès aux fichiers en lecture et en écriture plus intuitif, tout en accroissant les performances. De la même façon, les méthodes **GetFileInfo**, **GetDirectoryInfo** et **GetDriveInfo** fournissent des informations à propos des fichiers, des répertoires et des lecteurs. Pour plus d'informations, consultez Analyse des fichiers texte avec l'objet TextFieldParser.

Exploitation de réseau

L'My.Computer.Network, objet fournit des méthodes et des propriétés pour transférer et télécharger des fichiers, vérifier l'état de la connexion et déterminer si des ordinateurs distants sont disponibles. Pour plus d'informations, consultez Exécution d'opérations de réseau.

Ressources

L'My.Resources, objet vous permet d'accéder à des ressources, notamment des ressources audio, des ressources d'icône et des ressources localisées. Pour plus d'informations, consultez Accès aux ressources de l'application.

Vous pouvez ajouter et gérer des ressources à l'aide de la page **Ressources** du **Concepteur de projets**. Pour plus

d'informations, consultez Page Ressources, Concepteur de projets.

Paramètres utilisateur et paramètres d'application

L'My.Settings, objet permet de modifier des paramètres utilisateur et de les rendre persistants, ainsi que de lire des paramètres d'application. Pour plus d'informations, consultez Accès aux paramètres d'application.

Vous pouvez également gérer des paramètres d'application à l'aide de la page **Paramètres** du **Concepteur de projets**. Pour plus d'informations, consultez Page Paramètres, Concepteur de projets.

Accès aux données

Visual Basic 2005 inclut plusieurs nouvelles fonctionnalités qui vous aident à développer des applications qui accèdent aux données. L'Assistant Configuration de source de données simplifie la connexion de votre application à des données présentes dans des bases de données, des services Web et des objets créés par l'utilisateur.

Le nouveau Sources de données, fenêtre fournit un emplacement central pour consulter les données disponibles et associées à votre projet. Il réduit la complexité de la liaison de données en autorisant le glissement d'éléments de la fenêtre vers des formulaires en vue de créer des contrôles liés aux données.

Groupes de données de remplissage, exécution de requêtes et de procédures stockées ; autant d'opérations qui peuvent désormais être effectuées à l'aide du nouvel objet généré par Visual Studio, TableAdapter. La nouvelle fonctionnalité de données locale vous permet d'intégrer des fichiers de base de données Microsoft Access et Microsoft SQL Server Express directement à votre application. Pour une liste complète des nouvelles fonctionnalités, consultez Nouveautés des données.

Améliorations au moment du design

Modifier & Continuer

Modifier & Continuer vous permet d'apporter des modifications à votre application pendant que celle-ci s'exécute dans le cadre du débogueur sans l'arrêter ni la redémarrer. Les modifications que vous apportez en mode arrêt s'appliquent immédiatement lorsque vous reprenez l'exécution de votre application.

Pour plus d'informations, consultez Modifier & Continuer (Visual Basic).

Édition d'attributs dans la fenêtre Propriétés

Vous pouvez désormais modifier des attributs communs qui s'appliquent à vos classes et méthodes dans la fenêtre **Propriétés**, plutôt que de devoir les appliquer dans un code.

Pour plus d'informations, consultez Comment : modifier des attributs de code.

Filtrage IntelliSense

Vous pouvez désormais régler le niveau de détails fourni par IntelliSense dans l'Éditeur de code. Lorsque la fenêtre IntelliSense s'affiche, vous pouvez éliminer par filtrage les membres généralement les moins utilisés en cliquant sur l'onglet **Commun**, ou en affichant toutes les ressources disponibles en cliquant sur l'onglet **Tout**. Le réglage du niveau de détails facilite l'élimination des membres rares ou ésotériques superflus, ou leur inclusion lorsqu'ils sont nécessaires.

Pour plus d'informations, consultez Listes filtrées des opérations terminées dans Visual Basic.

IntelliSense par zone

IntelliSense par zone est une fonctionnalité IntelliSence Visual Basic. Elle affiche certains éléments dans la liste de saisie semi-automatique d'instructions dans une couleur distincte si ceux-ci ne disposent pas d'une autorisation suffisante pour s'exécuter dans la zone de sécurité spécifiée. Cette prise en charge au moment du design facilite le développement d'applications qui s'exécutent dans le cadre d'une approbation partielle.

Pour plus d'informations, consultez Options IntelliSense spécifiques à Visual Basic. Voir aussi Comment : définir une zone de sécurité pour une application ClickOnce.

Extraits de code IntelliSense

La bibliothèque d'extraits de code IntelliSense se compose de 380 éléments de code prédéveloppés. Vous pouvez ajouter ces routines réutilisables à votre code, puis les modifier à l'aide des pointeurs que le code contient. Pour accéder aux extraits de code disponibles, cliquez avec le bouton droit dans l'**éditeur de code**, puis cliquez sur **Insérer un extrait**. Ces extraits sont également accessibles par le biais de raccourcis clavier.

Les tâches de ces extraits de code vont de la création d'une exception personnalisée à l'envoi d'un courrier électronique, en passant par le dessin d'un cercle. Certains sont de simples extensions de structures de programmation communes. La bibliothèque est extensible ; vous pouvez créer des tâches de code adaptées aux besoins de votre entreprise, puis les ajouter à la bibliothèque. Vous pouvez également télécharger d'autres tâches de bibliothèque auprès de fournisseurs de logiciels tiers et

des sites de la communauté Visual Basic.

Pour plus d'informations, consultez Extraits de code IntelliSense Visual Basic et Comment : insérer des extraits de code dans votre code (Visual Basic).

Correction d'erreurs et avertissements

La **Correction d'erreur à compilation intelligente** permet à l'éditeur de code de suggérer des solutions aux erreurs et avertissements courants, et de vous laisser sélectionner la correction appropriée qui sera alors appliquée à votre code. Lorsqu'une erreur se produit et qu'un symbole se présente sous le côté droit du tilde, positionnez la souris sur le tilde pour transformer le symbole en un panneau de balises actives. Cliquez sur la balise active pour ouvrir la fenêtre du programme d'assistance **Options de correction d'erreurs**. Celle-ci fournit une description de l'erreur, suggère différentes manières possibles de la résoudre et, dans les cas appropriés, permet à l'utilisateur de prévisualiser la correction.

Pour plus d'informations, consultez Correction automatique de la compilation intelligente et Configuration d'avertissements en Visual Basic.

Assistant Exception

Par défaut, lorsque vous déboguez une application, l'**Assistant Exception** s'ouvre pour vous aider à traiter les exceptions non gérées. Si vous le souhaitez, vous pouvez modifier l'environnement afin que l'assistant s'affiche lorsque n'importe quelle exception se produit dans votre code. L'**Assistant Exception** répertorie le type de l'exception, le message spécifique à l'exception levée, une liste de conseils de dépannage et les actions qu'il est possible d'effectuer.

Pour plus d'informations, consultez Comment : corriger des erreurs d'exécution avec l'Assistant Exception.

Documentation XML

La documentation du code XML vous permet de documenter vos classes au sein même du code, et d'exposer cette documentation au format XML. La documentation XML résultante peut être exploitée et affichée de différentes manières. Elle vous permet de documenter votre code pendant que vous rédigez ce dernier.

Pour plus d'informations, consultez Comment : créer une documentation XML en Visual Basic.

Structure du document, fenêtre

La fenêtre **Structure du document** prend désormais en charge le mode Plan pour les Windows Forms en plus des pages Web ASP.NET et des pages HTML. Vous pouvez utiliser la fenêtre **Structure du document** pour naviguer parmi les contrôles sur les Windows Forms lorsque vous vous trouvez dans le mode **Design** de l'éditeur. Pour accéder à cette fenêtre, dans le menu **Affichage**, cliquez sur **Autres fenêtres**, puis sur **Structure du document**. Pour plus d'informations, consultez Structure du document, fenêtre

Concepteur de projets pour la gestion de projets, de paramètres et de ressources

Le **Concepteur de projets** constitue un emplacement centralisé pour gérer des propriétés, des paramètres et des ressources de projet. Il est accessible par le biais de la commande **Propriétés** du menu **Projet**.

Pour plus d'informations, consultez Introduction au Concepteur de projets.

Concepteur de paramètres

La page **Paramètres** du **Concepteur de projets** vous permet de spécifier les paramètres d'application d'un projet qui peuvent être de portée utilisateur ou de portée application. Cette fonctionnalité remplace les propriétés dynamiques. Pour plus d'informations, consultez Page Paramètres, Concepteur de projets.

Concepteur de ressources

Le **Concepteur de ressources.** est un outil d'interface utilisateur qui vous permet de gérer les ressources (telles que des chaînes, des images, des icônes, des sons et des fichiers) utilisées par un projet. Il crée des ressources fortement typées auxquelles vous pouvez accéder au moment de l'exécution à l'aide de l'My.Resources, objet. Pour plus d'informations sur l'accès aux ressources à l'aide de l'objet **My.Resources**, consultez Accès aux ressources de l'application.

La page **Ressources** du **Concepteur de projets** héberge une instance du **Concepteur de ressources** qui stocke et gère les ressources à un emplacement unique (Resources.resx). Pour plus d'informations, consultez Page Ressources, Concepteur de projets.

Déploiement ClickOnce

Le déploiement ClickOnce vous permet de publier des applications Windows et console à mise à jour automatique, qui peuvent être installées, mises à jour, et exécutées aussi facilement que des applications Web. Pour plus d'informations,

consultez Déploiement ClickOnce.

Les nouveaux onglets **Sécurité**, **Signature** et **Publier** du **Concepteur de projets** permettent de personnaliser votre déploiement ClickOnce. La nouvelle commande **Publier** du menu **Générer** (ou du menu contextuel de l'**Explorateur de solutions**) permet d'accéder à l'**Assistant Publication**, un outil qui vous guide étape par étape dans la publication de votre application. Pour plus d'informations, consultez Page Publier, Concepteur de projets, Page Signature, Concepteur de projets et Page Sécurité, Concepteur de projets.

Déploiement d'applications 64 bits

Les déploiements ClickOnce et Windows Installer prennent en charge l'installation sur des plateformes 64 bits. Pour plus d'informations, consultez Déploiement d'applications 64 bits.

Exemples

Visual Basic 2005 est livré avec plus de 60 exemples d'applications. Les nouvelles fonctionnalités couvertes sont notamment les suivantes :

- Déploiement ClickOnce
- Programmation avec l'objet **My**
- Améliorations du langage, notamment des génériques et des commentaires XML
- Accès aux données

Pour plus d'informations, consultez Exemples d'applications Visual Basic.

Mise à niveau de Visual Basic 6.0

Les outils de mise à niveau de Visual Basic 2005 présentent de nombreuses améliorations, notamment la capacité de mettre à niveau un grand nombre de contrôles ActiveX Visual Basic 6.0 vers leurs équivalents Visual Basic 2005.

Pour plus d'informations, consultez Nouveautés de la mise à niveau.

Voir aussi

Concepts

Nouveautés de Visual Studio 2005
Nouveautés dans les Windows Forms et les contrôles Windows Forms
Nouveautés dans ASP.NET
Nouveautés des projets Web
Nouveautés du développement Web pour Visual Studio
Nouveautés du déploiement

Nouveautés du langage Visual Basic

Visual Basic 2005 introduit de nouvelles fonctionnalités du langage, à savoir la continuation de boucle, la disponibilité de ressources garantie, les propriétés d'accès mixtes, les types de données non signés et nullables, la surcharge d'opérateur, les types partiels et génériques, les événements personnalisés et la vérification de la conformité Spécification CLS (Common Language Specification) (CLS).

Cette page répertorie les fonctionnalités nouvelles du langage Visual Basic par rapport à Visual Basic 2005. Pour obtenir un examen des fonctions et fonctionnalités ayant fait l'objet de modifications depuis les versions antérieures, consultez Modifications du langage pour les utilisateurs de Visual Basic 6.0.

Continue, instruction

Visual Basic fournit désormais une instruction **Continue** qui passe immédiatement à l'itération suivante d'une boucle **Do**, **For** ou **While**. Pour plus d'informations, consultez Comment : passer à l'itération suivante d'une boucle et Continue, instruction (Visual Basic).

Accès aux formulaires Visual Basic 6.0

Visual Basic vous permet désormais de faire référence à un formulaire défini, par exemple `Form1`, en utilisant son nom de classe au lieu d'en créer explicitement une instance. Pour plus d'informations, consultez Comment : accéder à un formulaire.

IsNot, opérateur

Visual Basic fournit désormais un opérateur **IsNot** qui vous permet d'éviter d'utiliser les opérateurs **Not** et **Is** dans un ordre compliqué. Pour plus d'informations, consultez Comment : déterminer si deux objets sont identiques et IsNot, opérateur.

Opérateur TryCast

Visual Basic fournit maintenant l'opérateur de conversion de type **TryCast** qui retourne **Nothing** si la tentative de conversion échoue, contrairement à **CType** et **DirectCast**, qui tous deux lèvent une erreur InvalidCastException. Pour plus d'informations, consultez TryCast.

Instruction Using

Visual Basic permet désormais à un bloc **Using...End Using** de garantir la disponibilité d'une ressource système lorsque votre code abandonne le bloc pour une raison donnée. Pour plus d'informations, consultez Comment : supprimer une ressource système et Using, instruction (Visual Basic).

Limite inférieure nulle explicite sur un tableau

Visual Basic permet désormais à une déclaration de tableau de spécifier la limite inférieure (0) de chaque dimension avec la limite supérieure. Pour plus d'informations, consultez Comment : spécifier une limite inférieure égale à zéro dans un tableau.

Propriétés avec les niveaux d'accès mixtes

Visual Basic vous permet désormais de déclarer une propriété avec différents niveaux d'accès différents sur ses procédures **Get** et **Set**. Pour plus d'informations, consultez Comment : déclarer une propriété avec des niveaux d'accès mixtes.

Types non signés

Visual Basic prend désormais en charge des types de données entiers non signés (**UShort**, **UInteger** et **ULong**), ainsi que le type **SByte** signé. Pour plus d'informations, consultez Comment : appeler une fonction Windows qui possède des types non signés, Comment : optimiser le stockage d'entiers positifs avec les types non signés, UInteger, type de données, ULong, type de données (Visual Basic), UShort, type de données (Visual Basic) et SByte, type de données (Visual Basic).

Types nullables

Visual Basic prend désormais en charge les extensions des types valeur qui peuvent adopter leurs valeurs normales ou une valeur nulle. Une valeur nulle permet d'indiquer qu'une variable n'a aucune valeur définie parce que les informations ne sont pas actuellement disponibles. Pour plus d'informations, consultez Types valeur et types référence et Types valeur dont la valeur n'est pas définie.

Surcharge d'opérateur

Visual Basic vous permet désormais d'affecter à un opérateur standard (tel que **+**, **&**, **Not** ou **Mod**) une classe ou une structure que vous avez définie. Pour plus d'informations, consultez Comment : utiliser une classe qui définit des opérateurs, Comment : définir un opérateur et Operator, instruction.

Séparation de code à l'aide des types partiels

Visual Basic fournit désormais un mécanisme permettant à l'environnement de développement intégré (IDE) de séparer le code généré du code que vous avez créé dans des fichiers sources distincts. Le plus souvent, vous vous limitez à la gestion du code que vous avez écrit. Pour plus d'informations, consultez Partial (Visual Basic).

Types génériques

Visual Basic prend désormais en charge les paramètres de type des classes génériques, des structures, des interfaces, des procédures et des délégués. Un argument de type correspondant spécifie, au moment de compilation, le type de données de l'un des éléments dans le type générique. Pour plus d'informations, consultez Types génériques en Visual Basic.

Événements personnalisés

Visual Basic vous permet désormais d'avoir un plus grand contrôle sur le comportement détaillé des événements. Vous pouvez déclarer des événements personnalisés en utilisant le mot clé **Custom** comme modificateur de l'instruction **Event**. Dans un événement personnalisé, vous spécifiez exactement ce qui se passe lorsque le code ajoute ou supprime un gestionnaire d'événements de l'événement ou lorsque le code déclenche ce dernier. Pour obtenir des exemples, consultez Event, instruction, Comment : déclarer des événements qui économisent l'utilisation de la mémoire et Comment : déclarer des événements qui évitent les blocages. Pour conserver la compatibilité descendante avec le code existant, le mot clé **Custom** n'est pas un mot clé réservé.

Options de vérification du compilateur

Visual Basic 2005 présente les nouvelles options de vérification du compilateur. Les options /nowarn et /warnaserror (Visual Basic) offrent un plus grand contrôle sur le mode de gestion des avertissements. Chacune de ces options du compilateur considère désormais une liste d'ID d'avertissement comme un paramètre optionnel pour spécifier les avertissements auxquels s'applique l'option.

- Vérification de la conformité CLS

Visual Basic génère désormais un avertissement pour chaque ligne de code qui contient une spécification ou une opération que la Spécification CLS (Common Language Specification) (CLS) ne prend pas en charge.

- Vérification des variables non initialisées

Visual Basic génère désormais un avertissement pour chaque variable potentiellement non initialisée. Une variable est dans cet état s'il existe au moins un chemin d'exécution possible qui n'assigne pas de valeur à la variable avant de l'utiliser.

Pour plus d'informations, consultez Configuration d'avertissements en Visual Basic.

Voir aussi

Référence

Liste des types de données

Concepts

Nouveautés de Visual Studio 2005
Écriture d'un code conforme CLS
Durée de vie d'un objet : création et destruction des objets

Autres ressources

Référence Visual Basic
Gestion des exceptions et des erreurs en Visual Basic

Nouveautés du compilateur Visual Basic

Visual Basic 2005 inclut de nouvelles options et d'autres modifications du compilateur.

Compilation conditionnelle

Visual Basic présente les nouvelles constantes du compilateur conditionnelles suivantes pour tous les projets :

- **TRACE**. Une valeur Boolean que vous pouvez définir pour spécifier que les méthodes de la classe **Trace** génèrent une sortie sur la fenêtre Sortie ;
- **VBC_VER**. Version Visual Basic, au format *major.minor*. Pour Visual Basic 2005, la valeur de **VBC_VER** est 8.0.

Pour plus d'informations, consultez Constantes de compilation conditionnelle

Options de ligne de commande par défaut dans Vbc.rsp

Le compilateur de ligne de commande Visual Basic stocke désormais les options de ligne de commande par défaut dans le fichier Vbc.rsp, dans le même répertoire que le compilateur de ligne de commande, Vbc.exe. Vous pouvez modifier Vbc.rsp pour reconfigurer les options de ligne de commande par défaut.

Nouvelles options de ligne de commande

Il existe sept nouvelles options du compilateur de ligne de commande :

- L'option /codepage spécifie quelle page de code utiliser lors de l'ouverture des fichiers sources.
- L'option /doc génère un fichier de documentation XML basé sur les commentaires de votre code.
- L'option /errorreport représente un moyen pratique de rapporter à Microsoft une erreur du compilateur Visual Basic interne.
- L'option /filealign spécifie la taille des sections dans votre fichier de sortie.
- L'option /noconfig spécifie que le compilateur ne doit pas référencer automatiquement les assemblys du .NET Framework couramment utilisés ou importer les espaces de noms **System** et **Microsoft.VisualBasic**.
- L'option /nostdlib supprime la référence automatique à l'assembly System.dll et à d'autres assemblys couramment utilisés.
- L'option /platform spécifie le processeur à cibler par le fichier de sortie, dans des situations où il est nécessaire de le spécifier explicitement.

/Nowarn et /warnaserror

Les options /nowarn et /warnaserror offrent plus de contrôle sur le mode de gestion des avertissements. Ces options du compilateur considèrent désormais une liste d'ID d'avertissement comme un paramètre optionnel pour spécifier les avertissements auxquels s'applique l'option.

Voir aussi

Concepts

Nouveautés du langage Visual Basic

Éditions Visual Basic

Cette rubrique présente et décrit les fonctionnalités incluses dans les différentes éditions de Visual Basic.

Modèles de projet Visual Basic

La meilleure façon de créer des projets consiste à utiliser des modèles Visual Basic ou des Assistants Application. Les modèles de projets fonctionnent en combinaison avec les bibliothèques et les infrastructures d'application pour créer des programmes de démarrage pour vous.

Le tableau suivant montre les modèles de projet Visual Basic qui sont disponibles dans les différentes éditions de Visual Studio.

Modèle	**Microsoft Visual Basic 2005 Express**	**Visual Studio 2005 Standard**	**Visual Studio 2005 Professional et au-dessus**
Site Web ASP.NET		X	X
Service Web ASP.NET		X	X
Site Web ASP.NET Crystal Reports			X
Bibliothèque de classes	X	X	X
Application console	X	X	X
Application Crystal Reports			X
Application Smart Device		X	X
Projet vide	X	X	X
Site Web vide.		X	X
Modèle Excel			X
Classeur Excel			X
Starter Kit Ma cinémathèque	X	X	X
Complément Outlook			X
Starter Kit de site Web personnel		X	X
Pocket PC 2003 : Bibliothèque de classes		X	X
Pocket PC 2003 : Bibliothèque de classes (1.0)		X	X
Pocket PC 2003 : Application console		X	X
Pocket PC 2003 : Application console (1.0)		X	X
Pocket PC 2003 : Bibliothèque de contrôles		X	X

Pocket PC 2003 : Application Smart Device		X	X
Pocket PC 2003 : Application Smart Device (1.0)		X	X
Pocket PC 2003 : Projet vide		X	X
Pocket PC 2003 : Projet vide (1.0)		X	X
Starter Kit Écran de veille	X	X	X
Projet SQL Server			X
Smartphone 2003 : Bibliothèque de classes (1.0)		X	X
Smartphone 2003 : Application console (1.0)		X	X
Smartphone 2003 : Application Smart Device (1.0)		X	X
Smartphone 2003 : Projet vide (1.0)		X	X
Projet de test			X
Bibliothèque de contrôles Web		X	X
Application Windows	X	X	X
Windows CE 5.0 : Bibliothèque de classes		X	X
Windows CE 5.0 : Application console		X	X
Windows CE 5.0 : Bibliothèque de contrôles		X	X
Windows CE 5.0 : Application Smart Device		X	X
Windows CE 5.0 : Projet vide		X	X
Bibliothèque de contrôles Windows		X	X
Service Windows			X
Document Word			X
Modèle Word			X

Pour plus d'informations sur les modèles et éléments de projet disponibles, consultez Modèles de projet par défaut dans Visual Studio. Pour plus d'informations sur les types de fichiers, consultez

Types et extensions de fichiers en Visual Basic, Visual C# et Visual J#.

Outils visuels

Les concepteurs suivants sont intégrés à Visual Basic.

Fonctionnalité	Description	Microsoft Visual Basic 2005 Express	Visual Studio 2005 Standard	Visual Studio 2005 Professional et au-dessus
Concepteur Web Forms	Fournit un ensemble d'outils visuels pour travailler avec des pages et des contrôles Web Forms ASP .NET.			X
Concepteur Windows Forms	Fournit un ensemble d'outils visuels pour travailler avec les formulaires et les contrôles Windows Forms.	X	X	X
Concepteur de composants	Fournit un ensemble d'outils visuels pour travailler avec des composants non visuels.			X
Concepteur de schémas XML (Concepteur XML)	Fournit un ensemble d'outils visuels pour travailler avec des schémas XML, des groupes de données ADO.NET et des documents XML.			X
Visual Database Tools	Fournit un jeu d'outils à utiliser avec les bases de données.	X	X	X
Affichage de classes	Affiche les symboles définis, référencés ou appelés dans l'application que vous développez		X	X
Crystal Reports	Fournit des outils permettant de créer des rapports pour des applications Windows et Web.			X
Visionneuse de rapports	Fournit des outils de création de rapports pour les applications personnalisées.		X	

Pour afficher la liste complète des fonctionnalités proposées dans les différentes éditions de Visual Studio, consultez Éditions de Visual Studio.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Référence

Paramètres de Visual Basic

Concepts

Notions fondamentales sur l'installation et la configuration

Éditions de Visual Database Tools

Types de projets dans Visual Basic Express

Autres ressources

Éditions de Visual Studio

Procédures pas à pas relatives au langage Visual Basic

Les procédures pas à pas fournissent des instructions étape par étape pour des scénarios courants et constituent un excellent point de départ pour se familiariser avec le produit ou avec un ensemble particulier de fonctionnalités.

Dans cette section

Procédure pas à pas : déclaration et déclenchement des événements

Explique comment les événements sont déclarés et déclenchés dans Visual Basic.

Gestion des événements

Explique comment gérer les événements en utilisant le mot clé **WithEvents** standard ou les nouveaux mots clés **AddHandler**/**RemoveHandler**.

Définition de classes

Décrit comment déclarer une classe et ses champs, propriétés, méthodes et événements.

Appel à des API Windows

Explique comment utiliser les instructions **Declare** et appeler des API Windows. Inclut des informations sur l'utilisation d'attributs pour contrôler le marshaling en vue de l'appel API et l'exposition d'un appel API sous forme d'une méthode de classe.

Création et implémentation d'interfaces

Explique comment les interfaces sont déclarées et implémentées dans Visual Basic.

Multithreading

Explique comment créer une application multithread qui recherche les occurrences d'un mot dans un fichier texte.

Gestion structurée des exceptions

Montre comment créer et gérer des programmes avec une gestion d'erreurs complète, et comment utiliser **Try...Catch...Finally**.

Création d'objets COM avec Visual Basic

Montre comment créer des objets COM dans Visual Basic, avec ou sans le modèle de classe COM.

Implémentation de l'héritage avec les objets COM

Montre comment utiliser Visual Basic 6.0 pour créer un objet COM contenant une classe, puis l'utiliser comme classe de base dans Visual Basic.

Manipulation de fichiers et de dossiers dans Visual Basic

Montre comment utiliser des fonctions Visual Basic pour déterminer des informations relatives à un fichier, rechercher une chaîne dans un fichier et écrire dans un fichier.

Manipulation de fichiers à l'aide de méthodes .NET Framework

Montre comment utiliser des méthodes .NET Framework pour déterminer des informations relatives à un fichier, rechercher une chaîne dans un fichier et écrire dans un fichier.

Rubriques connexes

Procédures pas à pas Visual Basic

Fournit des liens vers les procédures pas à pas Visual Basic qui abordent les applications distribuées, les données, les services Web, les Windows Forms, les Web Forms, la création de composants et de contrôles, les services Framework, la mise à niveau, l'internationalisation et l'accessibilité.

Ressources supplémentaires pour les programmeurs Visual Basic

Les sites Web et newsgroups suivants peuvent vous aider à résoudre les problèmes courants.

Ressources Microsoft

Sur le Web

Aide et support Microsoft

Permet d'accéder aux articles de la Base de connaissances, ainsi qu'aux téléchargements, mises à jour, Webcasts de support technique et autres services.

Centre pour développeurs Microsoft Visual Basic

Fournit des exemples de code, des informations de mise à jour, des téléchargements gratuits et du contenu technique.

Blog de l'équipe Microsoft Visual Basic

Fournit l'accès au blog de l'équipe Visual Basic.

Microsoft ASP.NET

Fournit des articles, des démonstrations, des aperçus d'outils et d'autres informations pour le développement Web dans Visual Basic.

GotDotNet pour Visual Basic

Contient des articles, des exemples et d'autres informations qui présentent un intérêt pour les développeurs Visual Basic.

Conversations, forums et groupes de discussion

Forums MSDN

Propose des forums de discussion sur le Web pour de nombreuses technologies Microsoft y compris Visual Basic et .NET Framework.

Groupes de discussion MSDN

Permet une expérience de newsgroup, pour communiquer avec des experts du monde entier au sein d'une communauté.

Conversations MSDN

Fournit des discussions à propos des produits ou technologies Microsoft. Chaque conversation est hébergée par un ou plusieurs experts Microsoft. Des transcriptions des conversations complétées sont disponibles.

Vidéos et Webcasts

Visual Basic au cinéma

Expose 101 films courts destinés à diffuser de nouvelles compétences Visual Basic à tout le monde, du débutant au développeur avancé.

Channel9

Fournit une diffusion continue de communauté, par le biais de vidéos, Wikis et forums.

Ressources tierces

Le site Web MSDN fournit des informations sur les sites et les newsgroups tiers intéressants. Pour obtenir la liste la plus récente des ressources disponibles, consultez le site Web de la communauté MSDN Visual Basic.

Sur le Web

Zone DevX Visual Basic

Présente des articles techniques détaillés à l'intention des développeurs Visual Basic qui migrent vers Visual Basic 2005.

devCity.NET

Permet d'accéder à plus de 5 000 articles, codes source, didacticiels Microsoft Visual Basic et ASP (Active Server Pages), ainsi qu'aux informations soumises par les utilisateurs, en vous inscrivant à devCity.NET.

Voir aussi

Autres ressources

Mise en route de Visual Basic
Interaction avec d'autres développeurs
Support technique et accessibilité

Comment faire en Visual Basic

Comment faire est votre passerelle vers les rubriques clés basées sur des tâches relatives à la programmation et au développement d'applications en Visual Basic. Les catégories essentielles concernant les opérations réalisables à l'aide de Visual Basic sont répertoriées dans cette rubrique. Les liens fournissent des pointeurs vers les principales pages d'aide basées sur des procédures.

Apprentissage du langage Visual Basic (Comment faire en Visual Basic)

Programmation orientée objet … Gérer des flux de contrôle … Travailler avec des variables … Gérer des erreurs et des exceptions … autres

Mise à niveau pour les utilisateurs de Visual Basic 6 (Comment faire en Visual Basic)

Mettre à niveau une application Visual Basic 6 … Connaître le .NET Framework … autres

Accès aux données (Comment faire en Visual Basic)

Mise en route … Charger des données … Valider des données … autres

Déploiement (Comment faire en Visual Basic)

Déployer avec ClickOnce … Utiliser des projets d'installation … Utiliser des projets de déploiement … autres

Applications Windows (Comment faire en Visual Basic)

Formulaires et contrôles … Données … Impression … autres

Applications Web (Comment faire en Visual Basic)

Développer un site Web … Formulaires et contrôles utilisateur Web … Déployer un site Web … autres

Services Web (Comment faire en Visual Basic)

Créer un service Web … Appeler un service Web … Déboguer un service Web ... autres

Programmation Office (Comment faire en Visual Basic)

Applications Excel … Applications Word … autres

Applications Smart Device (Comment faire en Visual Basic)

Utiliser des données … Utiliser des contrôles … Déployer des applications Smart Device … autres

Sécurité (Comment faire en Visual Basic)

Applications Web … Applications Windows …Composants … autres

Débogage et test (Comment faire en Visual Basic)

Déboguer des applications … Surveiller des applications … autres

Mise à niveau pour les utilisateurs de Visual Basic 6 (Comment faire en Visual Basic)

Cette page fournit des liens qui vous aideront à vous familiariser avec Visual Basic .NET si vous utilisiez précédemment Visual Basic 6. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Aide pour les utilisateurs de Visual Basic 6.0

Met en surbrillance les modifications apportées à différentes fonctionnalités depuis Visual Basic 6.0.

Nouveautés du langage Visual Basic

Décrit les fonctionnalités nouvelles et améliorées du langage, notamment l'héritage, les interfaces, la substitution, les membres partagés et la surcharge.

Procédure pas à pas : mise à niveau d'une application Visual Basic 6.0 vers la version actuelle de Visual Basic

Illustre la mise à niveau d'un exemple d'application graphique Visual Basic vers Visual Basic .NET En plus d'illustrer le processus de mise à niveau, cette procédure pas à pas met en évidence les différences qui existent entre l'architecture graphique de Visual Basic 6.0 et celle de Visual Basic .NET.

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Détaille différentes mesures que vous pouvez prendre pour optimiser votre mise à niveau avec une petite préparation.

Si vous suivez les recommandations de cette rubrique, vous pourrez réduire et, le cas échéant, supprimer les modifications exigées après la mise à niveau d'un projet vers Visual Basic .NET.

Points à considérer avant d'effectuer la mise à niveau

Décrit des zones dans lesquelles la compatibilité avec les versions antérieures est rompue.

Développement avec My

Décrit la fonctionnalité de Visual Basic conçue pour donner accès à des informations et des instances d'objet par défaut mises en relation avec l'application et son environnement d'exécution.

Ressources supplémentaires pour les programmeurs Visual Basic

Obtenir des informations sur les différents sites Visual Basic de Microsoft et de tiers.

Mettre à niveau une application Visual Basic 6

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Met en surbrillance les modifications apportées à différentes fonctionnalités depuis Visual Basic 6.0.

Procédure pas à pas : mise à niveau d'une application Visual Basic 6.0 vers la version actuelle de Visual Basic

Illustre la mise à niveau d'un exemple d'application graphique Visual Basic vers Visual Basic .NET En plus d'illustrer le processus de mise à niveau, cette procédure pas à pas met en évidence les différences qui existent entre l'architecture graphique de Visual Basic 6.0 et celle de Visual Basic .NET.

Mise à niveau de Visual Basic 6.0

Fournit des liens vers des rubriques d'aide qui montrent comment mettre à niveau vos applications et découvrir rapidement les modifications apportées dans Visual Basic 2005.

Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic

Décrit étape par étape la mise à niveau d'un projet Visual Basic 6 avec un Assistant facile à utiliser.

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Détaille différentes mesures que vous pouvez prendre pour optimiser votre mise à niveau avec une petite préparation.

Si vous suivez les recommandations de cette rubrique, vous pourrez réduire et, le cas échéant, supprimer les modifications exigées après la mise à niveau d'un projet vers Visual Basic .NET.

Points à considérer avant d'effectuer la mise à niveau

Décrit des zones dans lesquelles la compatibilité avec les versions antérieures est rompue.

COM Interop

Décrit l'interopérabilité COM dans le contexte d'applications Visual Basic.

Ressources supplémentaires pour les programmeurs Visual Basic

Obtenir des informations sur les différents sites Visual Basic de Microsoft et de tiers.

Travailler avec des données (pour les utilisateurs Visual Basic 6)
Accès aux données pour les utilisateurs de Visual Basic 6.0

Décrit les différences entre les technologies ADO et ADO.NET en termes de concept et de tâches.

Comparaison entre ADO.NET et ADO

Explique la technologie ADO.NET aux utilisateurs habitués à ADO.

Connaître le .NET Framework
Vue d'ensemble conceptuelle du .NET Framework

Explique les fonctionnalités du Common Language Runtime et de la bibliothèque de classes .NET Framework.

Visual Basic et le .NET Framework

Explique les notions fondamentales de la programmation .NET du point de vue de Visual Basic.

Introduction à la bibliothèque de classes .NET Framework dans Visual Studio

Explique comment utiliser des espaces de noms et des assemblys.

VBRun

Centre de ressources Visual Basic 6.0. Fournit des liens vers des téléchargements, des articles, des groupes de discussion et d'utilisateurs, ainsi que d'autres ressources. http://msdn.microsoft.com/VBRun/

Ressources supplémentaires pour les programmeurs Visual Basic

Obtenir des informations sur les différents sites Visual Basic de Microsoft et de tiers.

Voir aussi
Concepts
Comment faire en Visual Basic

Apprentissage du langage Visual Basic (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches de programmation Visual Basic largement utilisées. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Nouveautés de Visual Basic

Répertorie les fonctionnalités nouvelles et améliorées disponibles dans cette version de Visual Basic intégrée à Visual Studio 2005.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Fournit une liste des modifications, avec des liens vers les détails, dont le langage Visual Basic a fait l'objet depuis Visual Basic 6.0.

Programmation orientée objet avec Visual Basic 2005

Quand utiliser l'héritage

Explique dans quels cas il convient d'utiliser l'héritage au lieu des interfaces.

Éléments fondamentaux de l'héritage

Présente les modificateurs d'héritage, la substitution de méthodes et de propriétés, **MyClass** et **MyBase**.

Comment : définir un opérateur

Indique comment définir le comportement d'un opérateur standard (tel que *****, **<>** ou **And**) lorsqu'un des opérandes ou les deux est du type de votre classe ou structure.

Comment : accéder aux membres d'un objet

Explique comment lire, écrire ou appeler un membre d'un objet que vous avez créé.

Comment : définir plusieurs versions d'une procédure

Explique comment définir une procédure dans plusieurs versions en la surchargeant, à l'aide du même nom, mais d'une liste de paramètres différente pour chaque version.

Utiliser des propriétés

Comment : placer une valeur dans une propriété

Montre comment stocker une valeur dans une propriété.

Comment : obtenir une valeur d'une propriété

Montre comment récupérer la valeur stockée dans une propriété.

Comment : déclarer et appeler une propriété par défaut en Visual Basic

Montre comment déclarer et appeler une propriété à laquelle vous pouvez accéder sans utiliser son nom.

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Illustre la déclaration de propriétés avec des niveaux d'accès différents pour le stockage et la récupération.

Comment : créer une propriété

Montre comment créer une propriété.

Utiliser l'héritage

Comment : définir une classe qui utilise les membres d'une classe existante.

Montre un exemple de code pour la création d'une classe qui dérive d'une autre classe.

Comment : masquer une variable héritée

Illustre l'utilisation du mot clé **Shadows** pour masquer une variable héritée.

Comment : accéder à une variable masquée par une classe dérivée

Montre comment accéder à une variable de classe de base qui est masquée dans une classe dérivée.

Utiliser des procédures

Comment : créer une procédure

Explique comment déclarer une procédure pour exécuter une tâche nécessaire à plusieurs endroits dans votre code.

Comment : appeler une procédure qui retourne une valeur

Montre deux moyens d'appeler une procédure **Function** et d'utiliser la valeur qu'elle retourne.

Comment : appeler une procédure qui ne retourne pas de valeur

Décrit comment appeler une procédure **Sub** qui exécute une tâche, mais ne retourne pas de valeur au code appelant.

Comment : définir un paramètre pour une procédure

Décrit la définition d'une liste de paramètres pour accepter les valeurs passées à la procédure par le code appelant.

Comment : passer des arguments à une procédure

Montre comment passer une valeur à une procédure en fournissant un argument pour chacun de ses paramètres.

Gérer un flux de contrôle de programme

Comment : fractionner des grands segments de code en segments plus petits.

Explique comment utiliser des procédures pour effectuer une programmation structurée dans votre code

Comment : diviser et combiner des instructions dans le code

Décrit la rupture d'une instruction unique dans plusieurs lignes de code source et la combinaison de plusieurs instructions sur une ligne source.

Comment : ajouter un commentaire à votre code (Visual Basic)

Montre deux façons de placer des commentaires explicatifs dans votre code source.

Instructions conditionnelles

Comment : exécuter des instructions en fonction d'une ou de plusieurs conditions

Montre l'utilisation de constructions **If**...**Then**...**Else** pour exécuter différents blocs d'instructions selon diverses conditions.

Fonctions et sous-routines

Comment : créer une procédure

Explique comment déclarer une procédure pour exécuter une tâche nécessaire à plusieurs endroits dans votre code.

Comment : appeler une procédure qui retourne une valeur

Montre deux moyens d'appeler une procédure **Function** et d'utiliser la valeur qu'elle retourne.

Comment : appeler une procédure qui ne retourne pas de valeur

Décrit comment appeler une procédure **Sub** qui exécute une tâche, mais ne retourne pas de valeur au code appelant.

Comment : définir un paramètre pour une procédure

Décrit la définition d'une liste de paramètres pour accepter les valeurs passées à la procédure par le code appelant.

Comment : passer des arguments à une procédure

Montre comment passer une valeur à une procédure en fournissant un argument pour chacun de ses paramètres.

Instructions de boucle

Comment : passer à l'itération suivante d'une boucle

Explique comment quitter l'itération actuelle d'une boucle et passer à l'itération suivante.

Comment : exécuter plusieurs instructions à plusieurs reprises

Montre comment exécuter un bloc d'instructions un nombre défini de fois ou jusqu'à ce qu'une condition soit remplie.

Comment : parcourir une collection dans Visual Basic

Illustre une boucle **For Each**...**Next** qui exécute les mêmes instructions pour chaque élément d'une collection.

Comment : améliorer les performances d'une boucle

Décrit le choix du type de données le plus efficace pour la variable de contrôle d'une boucle.

Utiliser des collections, des génériques et des tableaux

Comment : définir des collections dans vos classes

Illustre la définition et l'utilisation d'un simple objet **Collection** Visual Basic.

Gestion de vos objets avec les collections

Décrit les critères à prendre en compte lors du choix du type de collection pour le regroupement d'objets connexes.

Comment : ajouter, supprimer et récupérer des éléments d'une collection

Décrit les opérations de base de l'ajout d'un nouvel élément, de la suppression d'un élément existant et de la récupération d'un élément.

Comment : créer une collection d'objets

Décrit les collections Visual Basic et les collections génériques, la manière de les créer, ainsi que la manière de les parcourir.

Collections

Comment : créer un tableau d'objets

Explique l'utilisation de tableaux comme une autre moyen de regrouper des objets connexes.

Dépannage des collections

Répertorie les problèmes couramment rencontrés avec les collections et indique comment les gérer.

Génériques

Comment : utiliser une classe générique

Montre comment utiliser une classe qui prend un ou plusieurs paramètres de type.

Comment : passer un tableau à une procédure ou propriété

Décrit comment passer un tableau de valeurs à une procédure ou propriété à l'aide de la liste d'arguments.

Comment : définir une classe qui fournisse des fonctionnalités identiques pour différents types de données

Montre comment définir une classe unique qui s'adapte aux différents types de données pour exécuter les mêmes fonctionnalités.

Tableaux

Comment : créer un tableau

Montre deux moyens de créer un objet tableau et de l'assigner à une variable tableau.

Comment : initialiser une variable tableau

Explique comment stocker un tableau dans une variable tableau, et comment définir sa longueur ainsi que la valeur de ses éléments.

Comment : initialiser un tableau en escalier

Montre quatre façons de stocker un tableau de tableaux, ou un tableau en escalier, dans une variable, et comment définir sa longueur ainsi que la valeur de ses éléments.

Comment : initialiser un tableau multidimensionnel

Montre quatre façons de stocker un tableau de plusieurs dimensions dans une variable, et comment définir sa longueur ainsi que la valeur de ses éléments.

Comment : placer une valeur dans un tableau

Décrit comment stocker une valeur dans un élément de tableau.

Comment : trier un tableau dans Visual Basic

Explique comment trier par ordre alphabétique les éléments d'un tableau.

Comment : inverser le contenu d'un tableau dans Visual Basic

Montre comment inverser l'ordre des éléments d'un tableau.

Comment : spécifier une limite inférieure égale à zéro dans un tableau

Décrit comment rendre votre code plus lisible en déclarant la limite inférieure d'un tableau comme étant égale à zéro.

Travailler avec des variables

Comment : créer une variable

Décrit l'instruction **Dim** et les différents mots clés et clauses que vous pouvez utiliser pour créer une variable.

Comment : créer un objet

Décrit les variables objet et la manière de créer une instance d'une classe.

Comment : placer des données dans et en dehors d'une variable

Décrit comment stocker et récupérer la valeur d'une variable.

Déclarer et initialiser des variables

Comment : déclarer une variable objet et lui affecter un objet dans Visual Basic

Décrit le type de données **Object** et la manière lui assigner un objet de tout type.

Comment : stocker plusieurs valeurs dans une variable

Décrit les types de données composites tels que les structures, les tableaux et les classes qui peuvent contenir plusieurs valeurs.

Comment : stocker des valeurs True et False dans une variable

Décrit le type de données **Boolean** et la manière de déclarer des variables pour contenir des valeurs logiques.

Comment : augmenter la durée de vie d'une variable

Décrit le mot clé **Static** et la manière de l'utiliser pour faire en sorte qu'une variable survive à la disparition de son élément contenant.

Comment : déterminer si deux objets sont liés

Illustre l'utilisation de la méthode **GetType** pour déterminer si un objet hérite d'un autre.

Contrôler la portée des variables

Comment : contrôler la portée d'une variable

Décrit les niveaux de portée et la manière de les utiliser pour contrôler le code qui peut faire référence à une variable.

Comment : contrôler la disponibilité d'une variable

Explique les différents niveaux d'accès et la manière d'en assigner un à une variable pour contrôler le code qui peut la lire ou l'écrire.

Comment : masquer une variable portant le même nom que votre variable

Montre deux façons d'utiliser l'occultation pour masquer une variable avec une autre du même nom.

Interagir avec des types de données

Types de données en Visual Basic

Décrit comment assigner un type de données à différents types d'éléments de programmation.

Caractères et chaînes

Comment : stocker des caractères dans une variable

Décrit les types de données **Char** et **String** et la manière de déclarer des variables pour contenir des valeurs de caractère.

Comment : faire correspondre une chaîne à un modèle

Montre comment utiliser l'opérateur **Like** pour faire correspondre un caractère contenu dans une chaîne avec une série de jeux de caractères.

Comment : supprimer des parties d'une chaîne (Visual Basic)

Montre comment supprimer toutes les occurrences d'une sous-chaîne d'une chaîne.

Valeurs numériques

Comment : stocker des entiers dans une variable

Décrit les types de données **SByte**, **Short**, **Integer** et **Long** et la manière de déclarer des variables pour contenir des valeurs d'entiers signés.

Comment : stocker les chiffres les plus significatifs dans une variable

Décrit le type de données **Decimal** et la manière de déclarer des variables pour contenir jusqu'à 29 chiffres significatifs.

Comment : stocker le plus grand nombre possible dans une variable

Décrit les types de données **ULong**, **Decimal**, **Single** et **Double** et la manière de déclarer des variables pour contenir des valeurs très importantes.

Comment : stocker des fractions dans une variable

Décrit les types de données **Single** et **Double** et la manière de déclarer des variables pour contenir des valeurs fractionnelles.

Conversion de types de données

Comment : convertir un objet en un autre type dans Visual Basic

Décrit les conversions de types du type de données **Object** vers un type de données plus spécifique.

Date et heure

Comment : stocker des valeurs de date et d'heure dans une variable

Décrit le type de données **Date** et la manière de déclarer des variables pour contenir des valeurs de date et d'heure.

Types non signés

Comment : optimiser le stockage d'entiers positifs avec les types non signés

Décrit le type de données **UInteger** et la manière de déclarer des variables pour contenir des valeurs entières positives avec un rendement optimum.

Comment : appeler une fonction Windows qui possède des types non signés

Décrit les types de données **Byte**, **UShort**, **UInteger** et **ULong** et la manière d'interagir avec les fonctions qui utilisent des types non signés.

Devise

Comment : stocker des valeurs monétaires dans une variable.

Décrit le type de données **Decimal** et la manière de déclarer des variables pour contenir des valeurs monétaires.

Programmer à l'aide d'événements

Comment : appeler un gestionnaire d'événements en Visual Basic

Montre comment définir un événement et un gestionnaire d'événements, et comment utiliser l'instruction **AddHandler** pour les associer.

Comment : créer des gestionnaires d'événements dans l'éditeur de code Visual Basic

Explique comment utiliser l'éditeur de code de Visual Basic pour écrire du code de réponse aux événements.

Comment : déclencher un événement (Visual Basic)

Montre comment un événement et utilise l'instruction **RaiseEvent** pour le déclencher.

Procédure pas à pas : déclaration et déclenchement des événements

Décrit étape par étape le processus de déclaration et de déclenchement d'événements pour une classe.

Procédure pas à pas : gestion des événements

Montre comment écrire une procédure gestionnaire d'événements.

Gérer des erreurs et des exceptions

Comment : extraire des informations à partir d'un objet Error

Montre comment obtenir des informations des propriétés de l'objet **Err**.

Comment : conserver le contrôle si une erreur se produit

Montre comment exécuter un bloc d'instructions particulier si une exception spécifiée se produit lors de l'exécution du code.

Procédure pas à pas : gestion structurée des exceptions

Fournit une procédure pas à pas pour créer une application simple et insérer un code de gestion des exceptions.

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Montre comment utiliser un bloc **Try**...**Catch** pour tester une section de code.

Exemples

Exemples de langages Visual Basic

Ces exemples illustrent des concepts du langage Visual Basic.

Voir aussi

Concepts

Comment faire en Visual Basic

Accès aux données (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches de données Visual Basic largement utilisées. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Nouveautés des données

Fournit des informations et des liens vers les nouvelles fonctionnalités de données destinées aux applications de données clientes.

Mise en route de l'accès aux données

Fournit des liens vers les rubriques relatives à la création d'applications qui exploitent des données, à l'aide de Visual Studio.

Création d'applications de données avec Visual Studio

Décrit le processus de création d'une application de données dans Visual Studio.

Procédure pas à pas : création d'une application de données simple

Fournit des instructions pas à pas pour créer une application d'accès aux données de base.

Accès aux données pour les utilisateurs de Visual Basic 6.0

Décrit les modifications apportées à l'accès aux données et aux outils de données.

ADO.NET

Fournit des liens vers les rubriques qui expliquent les fonctionnalités d'accès aux données dans le .NET Framework.

Vue d'ensemble des données locales

Décrit les fonctionnalités proposées par Visual Studio pour l'utilisation de bases de données SQL Server Express et Access.

Liaison d'objets dans Visual Studio

Décrit les fonctionnalités proposées par Visual Studio pour la liaison de données à des objets.

Accès aux données ASP.NET (Visual Studio)

Fournit des liens vers des informations relatives à l'utilisation de données sur des pages Web ASP.NET.

Données dans les solutions Office

Contient des liens vers des rubriques qui expliquent le mode de fonctionnement des données dans les solutions Office.

Données dans les projets smart device managés

Décrit la manière de gérer les données des périphériques dans Visual Studio.

Affichage des données dans les Windows Forms

Vue d'ensemble de l'affichage des données

Décrit le processus d'affichage des données dans les Windows Forms.

Sources de données, fenêtre

Décrit comment créer des formulaires liés aux données en faisant glisser des éléments depuis la fenêtre Sources de données.

Comment : lier des données à des contrôles existants

Décrit comment lier un contrôle existant à un champ de données spécifique.

Comment : afficher des données dans un contrôle DataGridView Windows Forms

Décrit comment présenter des données dans un contrôle DataGridView dans un Windows Form.

Comment : afficher des données dans les contrôles Windows Forms individuels

Décrit comment présenter des données dans des contrôles individuels dans un Windows Form.

Comment : afficher des données connexes dans une application Windows

Décrit comment présenter des données issues de tables connexes dans un groupe de données dans un Windows Form.

Comment : ajouter une requête paramétrée à un formulaire dans une application Windows

Décrit comment modifier un Windows Form existant pour intégrer des contrôles aux paramètres d'entrée et exécuter une requête.

Comment : définir le contrôle à créer lors d'une opération de glisser-déplacer à partir de la fenêtre Sources de données

Décrit comment définir le contrôle qui sera créé lors du glissement d'éléments depuis la fenêtre Sources de données vers un Windows Form.

Comment : ajouter des contrôles personnalisés à la fenêtre Sources de données

Décrit comment modifier la liste de contrôles disponible pour les éléments dans la fenêtre Sources de données, et comment ajouter des contrôles personnalisés à la liste.

Comment : ouvrir la fenêtre Sources de données

Décrit comment ouvrir la fenêtre Sources de données dans l'IDE Visual Studio.

Procédure pas à pas : affichage de données sur un formulaire dans une application Windows

Fournit des détails pas à pas sur l'interrogation de données provenant d'une base de données et leur affichage sur un Windows Form.

Procédure pas à pas : affichage de données connexes sur un formulaire dans une application Windows

Fournit des instructions pas à pas pour afficher des données issues de deux tables connexes dans un Windows Form.

Procédure pas à pas : création d'un formulaire pour rechercher des données dans une application Windows

Fournit des instructions pas à pas pour créer un Windows Form qui recherche des enregistrements dans une base de données selon des valeurs entrées par un utilisateur.

Procédure pas à pas : création d'une table de correspondance

Fournit des instructions pas à pas pour afficher des données issues d'une table selon les données sélectionnées dans une autre.

Procédure pas à pas : passage de données entre formulaires dans une application Windows

Fournit des instructions pas à pas pour passer des valeurs d'un formulaire à un autre dans une application.

Procédure pas à pas : création d'un contrôle utilisateur prenant en charge la liaison de données simple

Fournit des instructions pas à pas pour créer un contrôle personnalisé avec des attributs spécifiques à l'exploitation de la liaison de données des Windows Forms.

Connecter des applications à des données

Vue d'ensemble de la connexion aux données dans Visual Studio

Fournit des informations sur la connexion de votre application aux données à l'aide des outils disponibles au moment du design, et sur les objets de connexion ADO.NET, à l'aide de Visual Studio.

Assistant Configuration de source de données

Fournit des informations sur la connexion aux données, et sur la création de sources de données, en exécutant l'**Assistant Configuration de source de données**.

Comment : enregistrer une chaîne de connexion

Décrit comment enregistrer des informations de connexion dans votre application.

Comment : modifier une chaîne de connexion

Décrit comment modifier des informations de connexion existantes, précédemment enregistrées dans votre application.

Comment : établir une connexion à des données dans un objet

Décrit comment établir une liaison de données à des valeurs de propriété présentes dans des objets à l'aide de l'**Assistant Configuration de source de données**.

Comment : établir une connexion à des données d'une base de données

Décrit comment créer une connexion entre votre application et une base de données à l'aide de l'**Assistant Configuration**

de source de données.

Comment : établir une connexion à des données dans un service Web

Décrit comment créer une connexion entre votre application et les données retournées issues d'un service Web à l'aide de l'**Assistant Configuration de source de données**.

Comment : établir une connexion à des données d'une base de données Access

Décrit comment créer une connexion entre votre application et une base de données Access à l'aide de l'**Assistant Configuration de source de données**.

Comment : établir une connexion aux données d'une base de données SQL Server Express

Décrit comment créer une connexion entre votre application et une base de données SQL Express à l'aide de l'**Assistant Configuration de source de données**.

Comment : créer des connexions à des bases de données SQL Server

Décrit comment créer des chaînes de connexion à des bases de données SQL Server.

Comment : créer des connexions à des bases de données Access

Décrit comment créer des chaînes de connexion à des bases de données Access.

Comment : créer des connexions à des bases de données Oracle

Décrit comment créer des chaînes de connexion à des bases de données Oracle.

Procédure pas à pas : connexion aux données dans une base de données

Fournit des instructions pas à pas pour créer une connexion entre votre application et une base de données.

Procédure pas à pas : connexion aux données dans une base de données SQL Server Express

Fournit des instructions pas à pas pour créer une connexion entre votre application et une base de données SQL Express.

Procédure pas à pas : connexion aux données dans un service Web

Fournit des instructions pas à pas pour créer une connexion entre votre application et un service Web.

Procédure pas à pas : connexion aux données dans des objets

Fournit des instructions pas à pas pour créer une connexion entre votre application et les propriétés d'un objet.

Procédure pas à pas : connexion aux données dans une base de données Access

Fournit des détails pas à pas pour créer une connexion entre votre application et un fichier de base de données Access.

Connexion à des sources de données

Décrit l'objet de connexion ADO.NET et son utilisation pour se connecter à une source de données.

Modifier des données

Comment : ajouter des lignes à un DataTable

Fournit les étapes pour créer des objets DataRow et les ajouter à une table de données.

Comment : modifier des lignes dans un DataTable

Fournit les étapes pour modifier une ligne de données existante dans une table de données.

Comment : supprimer des lignes d'un DataTable

Fournit les étapes pour supprimer des lignes de données spécifiques dans une table de données.

Comment : désactiver les contraintes pendant le remplissage d'un groupe de données

Fournit des détails pour désactiver temporairement le contrôle de contrainte dans un groupe de données.

Comment : valider des modifications dans un groupe de données

Fournit des informations et des détails pour accepter des modifications apportées aux données dans un groupe de données.

Comment : rechercher les lignes modifiées

Fournit les étapes pour déterminer si des données ont été modifiées dans un groupe de données.

Comment : récupérer des lignes modifiées

Fournit des détails pour retourner des lignes de données qui ont été modifiées depuis la dernière acceptation de modifications.

Comment : trouver une ligne spécifique dans un DataTable

Fournit des détails pour rechercher des données dans une table de données au moyen d'une valeur de clé primaire ou d'une valeur de colonne.

Comment : obtenir des versions spécifiques d'un DataRow

Fournit des détails pour retourner des lignes de données spécifiques à l'aide de l'énumération DataRowVersion.

Comment : trouver des lignes contenant des erreurs

Fournit des détails pour rechercher des données dans une table de données marquée avec une erreur.

Enregistrer des données

Vue d'ensemble de l'enregistrement de données

Fournit des liens vers des rubriques qui expliquent comment envoyer des données mises à jour d'une application vers la base de données.

Comment : enregistrer les modifications apportées à un groupe de données dans une base de données

Fournit des détails sur le renvoi des données mises à jour vers une base de données à l'aide de TableAdapters et de DataAdapters.

Comment : mettre à jour les données à l'aide d'un TableAdapter

Fournit des détails sur le renvoi des données mises à jour vers une base de données à l'aide de TableAdapters.

Comment : insérer de nouveaux enregistrements dans une base de données

Fournit des détails sur la création de nouveaux enregistrements dans une base de données.

Comment : mettre à jour les enregistrements dans une base de données

Fournit des détails sur le renvoi d'enregistrements modifiés à la base de données.

Comment : supprimer les enregistrements dans une base de données

Fournit des détails sur la suppression d'enregistrements à partir d'une base de données.

Procédure pas à pas : enregistrement de données dans une base de données (table unique)

Fournit des détails pas à pas pour enregistrer des données modifiées dans une table de données dans la base de données sous-jacente.

Procédure pas à pas : enregistrement de données dans une base de données (plusieurs tables)

Fournit des détails pas à pas pour enregistrer des données modifiées dans une deux de données connexes dans la base de données sous-jacente.

Procédure pas à pas : enregistrement de données dans une transaction

Fournit des détails pas à pas sur l'enregistrement de données à l'aide de l'espace de noms System.Transactions.

Utiliser des TableAdapters

Vue d'ensemble de TableAdapter

Explique ce que sont les TableAdapters et énumère les outils disponibles pour les créer.

Assistant Configuration de requêtes TableAdapter

Décrit comment exécuter l'Assistant et fournit des détails pour chaque écran de celui-ci.

Comment : créer des TableAdapters

Fournit les étapes pour créer un nouveau TableAdapter.

Comment : modifier des TableAdapters

Fournit les étapes pour modifier des TableAdapters existants.

Comment : créer des requêtes TableAdapter

Fournit les étapes pour ajouter une requête à un TableAdapter existant.

Comment : modifier des requêtes TableAdapter

Fournit les étapes pour modifier des requêtes TableAdapter existantes.

Utilisation de valeurs null dans des TableAdapters

Fournit la procédure d'activation des valeurs null dans les paramètres TableAdapter.

Comment : afficher les requêtes dans un TableAdapter

Explique comment consulter les requêtes existantes dans un TableAdapter.

Comment : créer des requêtes TableAdapter paramétrées

Fournit les étapes pour créer des requêtes TableAdapter qui acceptent des paramètres.

Comment : ajouter des requêtes globales à un groupe de données

Fournit les étapes pour créer un TableAdapter avec une requête qui renvoie une valeur unique.

Procédure pas à pas : création d'un TableAdapter avec plusieurs requêtes

Fournit des instructions pas à pas pour créer un TableAdapter et lui ajouter une requête.

Rapports d'affichage et de design

ReportViewer Web Server and Windows Forms Controls

Décrit deux nouveaux contrôles ReportViewer que vous pouvez utiliser pour incorporer des rapports dans une application Windows ou Web.

Creating Reports (avec Crystal Reports)

Fournit des informations détaillées sur l'utilisation du concepteur Crystal Reports incorporé dans Visual Studio.

Échantillons de données

Exemples de données

Ces échantillons illustrent l'accès aux données.

Extraire des données

Vue d'ensemble du remplissage des groupes de données et de l'interrogation des données

Décrit comment charger des données dans des groupes de données, comment exécuter des procédures stockées et comment appliquer des requêtes à une base de données.

Comment : remplir de données un groupe de données

Fournit des détails pour charger des données dans des groupes de données à l'aide de TableAdapters et de DataAdapters.

Comment : créer et exécuter une instruction SQL qui retourne des lignes

Fournit des détails pour créer et exécuter les instructions SQL qui retournent des lignes, à l'aide de requêtes TableAdapter et d'objets Command.

Comment : créer et exécuter une instruction SQL qui retourne une seule valeur

Fournit des détails pour créer et exécuter les instructions SQL qui retournent des valeurs uniques, à l'aide de requêtes TableAdapter et d'objets Command.

Comment : créer et exécuter une instruction SQL qui ne retourne aucune valeur

Fournit des détails pour créer et exécuter les instructions SQL qui ne retournent aucune valeur, à l'aide de requêtes TableAdapter et d'objets Command.

Comment : exécuter une procédure stockée qui retourne des lignes

Fournit des détails pour exécuter des procédures stockées qui retournent des lignes, à l'aide de requêtes TableAdapter et d'objets Command.

Comment : exécuter une procédure stockée qui retourne une seule valeur

Fournit des détails pour exécuter des procédures stockées qui retournent des valeurs uniques, à l'aide de requêtes

TableAdapter et d'objets Command.

Comment : exécuter une procédure stockée qui ne retourne aucune valeur

Fournit des détails pour exécuter des procédures stockées qui ne retournent aucune valeur, à l'aide de requêtes TableAdapter et d'objets Command.

Comment : définir et obtenir des paramètres pour des objets de commande

Fournit des détails pour affecter des valeurs à des paramètres dans les requêtes et les procédures stockées, et pour lire des valeurs dans des paramètres retournés par des commandes exécutées.

Valider des données

Vue d'ensemble de la validation des données

Présente les événements pouvant être utilisés pour analyser les données avant leur validation dans le groupe de données.

Comment : valider les données au cours des modifications de colonnes

Décrit comment valider des données au cours de l'événement ColumnChanging.

Comment : valider les données au cours des modifications de lignes

Décrit comment valider des données au cours de l'événement RowChanging.

Procédure pas à pas : ajout d'une validation à un groupe de données

Fournit des informations pas à pas pour ajouter un code de validation à un groupe de données.

Travailler avec les groupes de données

Comment : créer un groupe de données typé

Explique comment créer un groupe de données typé à l'aide d'outils de design dans Visual Studio.

Comment : étendre les fonctionnalités d'un groupe de données

Explique comment et où placer un code lors de l'extension de la fonctionnalité d'un groupe de données typé.

Comment : créer des DataTables

Fournit les étapes pour créer un nouveau DataTable à l'aide du **Concepteur de DataSet**.

Comment : ajouter des colonnes à un DataTable

Fournit les étapes pour créer un nouveau DataColumn dans un **DataTable** existant.

Comment : définir le type de données d'un DataColumn

Explique comment définir ou modifier la propriété de type de données d'un **DataColumn**.

Comment : modifier la légende d'un DataColumn

Explique comment définir la légende d'un **DataColumn** en lui attribuant un nom de colonne ou autre chose.

Comment : définir la valeur par défaut d'un DataColumn

Fournit des détails pour définir la valeur initiale d'un nouveau **DataColumn**.

Comment : restreindre une colonne de données pour contenir des valeurs uniques

Explique comment définir un **DataColumn** afin qu'il ne puisse pas contenir de valeurs en double.

Comment : définir une colonne de données comme clé primaire

Explique comment définir un **DataColumn** en tant que clé primaire dans un **DataTable**.

Comment : créer des colonnes de données qui affichent des expressions

Explique comment définir un **DataColumn** pour qu'il affiche une valeur calculée.

Comment : créer des DataRelations avec le Concepteur de DataSet

Expliquer comment utiliser le **Concepteur de DataSet** pour ajouter un objet DataRelation à un groupe de données.

Procédure pas à pas : création d'un DataTable dans le Concepteur de DataSet

Fournit des instructions pas à pas pour créer un **DataTable** et définir les **DataColumn** qui compose sa structure.

Procédure pas à pas : création d'un groupe de données avec le Concepteur de DataSet

Fournit des instructions pas à pas pour créer un groupe de données typé sans l'aide de l'**Assistant Configuration de source de données**.

Gérer des conflits et des utilisateurs multiples

Introduction à l'accès concurrentiel aux données dans ADO.NET

Présente les différentes méthodes de contrôle d'accès concurrentiel.

Comment : gérer les erreurs d'accès concurrentiel

Décrit comment utiliser l'objet **DBConcurrencyException** pour identifier les exceptions d'accès concurrentiel et l'enregistrement ayant provoqué l'erreur.

Procédure pas à pas : gestion d'une exception d'accès concurrentiel

Vous fournit des instructions pas à pas sur la façon d'identifier et de résoudre une erreur d'accès concurrentiel.

Travailler avec SQL Server 2005

Projets SQL Server

Fournit une vue d'ensemble de la nature des projets SQL Server, et des différents éléments qui peuvent être créés au sein de ces projets.

Avantages de l'utilisation de code managé pour créer des objets de base de données

Fournit une brève explication des avantages de l'utilisation d'un langage .NET pour créer des objets de base de données SQL Server.

Attributs pour les projets SQL Server et les objets de base de données

Fournit des informations sur les différents attributs utilisés dans les projets SQL Server.

Comment : créer un projet SQL Server

Fournit des étapes détaillées pour créer un nouveau projet SQL Server.

Comment : déployer des éléments de projet SQL Server sur un serveur SQL Server

Fournit des étapes détaillées pour déployer un projet SQL Server sur la base de données.

Comment : créer et exécuter une procédure stockée SQL Server CLR

Fournit des étapes détaillées pour créer une procédure stockée en code managé.

Comment : créer et exécuter un agrégat SQL Server CLR

Fournit des étapes détaillées pour créer un agrégat en code managé.

Comment : créer et exécuter un déclencheur SQL Server CLR

Fournit des étapes détaillées pour créer un déclencheur en code managé.

Comment : créer et exécuter une fonction définie par l'utilisateur SQL Server CLR

Fournit des étapes détaillées pour créer une fonction définie par l'utilisateur en code managé.

Comment : créer et exécuter un type défini par l'utilisateur SQL Server CLR

Fournit des étapes détaillées pour créer un type défini par l'utilisateur en code managé.

Débogage de SQL Server 2005

Comment : déboguer une procédure stockée SQL CLR

Montre comment déboguer des procédures stockées sur un serveur SQL Server.

Procédure pas à pas : débogage d'un déclencheur SQL CLR

Montre comment effectuer un pas à pas détaillé d'une procédure stockée dans un déclencheur lorsqu'il se déclenche.

Procédure pas à pas : débogage d'un agrégat SQL CLR défini par l'utilisateur

Montre comment exécuter la fonction Pas à pas détaillé d'un agrégat défini par l'utilisateur.

Procédure pas à pas : débogage d'une fonction scalaire SQL CLR définie par l'utilisateur

Montre comment exécuter la fonction Pas à pas détaillé d'une fonction scalaire définie par l'utilisateur.

Procédure pas à pas : débogage d'une fonction table SQL CLR définie par l'utilisateur

Montre comment exécuter la fonction Pas à pas détaillée d'une fonction table définie par l'utilisateur.

Procédure pas à pas : débogage d'un type SQL CLR défini par l'utilisateur

Montre comment exécuter la fonction Pas à pas détaillé d'un type défini par l'utilisateur.

Débogage d'une base de données SQL CLR

Décrit comment déboguer des objets de base de données CLR.

Procédure pas à pas : débogage d'une procédure stockée T-SQL

Montre comment déboguer des procédures stockées sur un serveur SQL Server.

Procédure pas à pas : débogage d'un déclencheur T-SQL

Montre comment effectuer un pas à pas détaillé d'une procédure stockée dans un déclencheur lorsqu'il se déclenche.

Procédure pas à pas : débogage d'une fonction T-SQL définie par l'utilisateur

Montre comment exécuter la fonction Pas à pas détaillé d'une fonction définie par l'utilisateur à partir d'une procédure stockée.

Débogage d'une base de données T-SQL

Décrit comment déboguer des objets de base de données T-SQL.

Procédures pas à pas relatives aux données

Procédure pas à pas : ajout d'une validation à un groupe de données

Fournit des informations pas à pas pour ajouter un code de validation à un groupe de données.

Procédure pas à pas : connexion aux données dans une base de données

Fournit des instructions pas à pas pour créer une connexion entre votre application et une base de données.

Procédure pas à pas : connexion aux données dans une base de données SQL Server Express

Fournit des instructions pas à pas pour créer une connexion entre votre application et une base de données SQL Express.

Procédure pas à pas : connexion aux données dans un service Web

Fournit des instructions pas à pas pour créer une connexion entre votre application et un service Web.

Procédure pas à pas : connexion aux données dans une base de données Access

Fournit des détails pas à pas pour créer une connexion entre votre application et un fichier de base de données Access.

Procédure pas à pas : connexion aux données dans des objets

Fournit des instructions pas à pas pour créer une connexion entre votre application et les propriétés d'un objet.

Procédure pas à pas : création d'un groupe de données avec le Concepteur de DataSet

Fournit des instructions pas à pas pour créer un groupe de données typé sans l'aide de l'Assistant Configuration de source de données.

Procédure pas à pas : création d'un DataTable dans le Concepteur de DataSet

Fournit des instructions pas à pas pour créer un T:System.Data.DataTable et définir le T:System.Data.DataColumns qui constitue sa structure.

Procédure pas à pas : création d'une table de correspondance

Fournit des instructions pas à pas pour afficher des données issues d'une table selon les données sélectionnées dans une autre.

Procédure pas à pas : création d'une relation entre des tables de données

Fournit des instructions pas à pas pour créer deux tables de données avec le Concepteur de DataSet, et établir une relation entre celles-ci.

Procédure pas à pas : création d'un TableAdapter avec plusieurs requêtes

Fournit des instructions pas à pas pour créer un TableAdapter et lui ajouter une requête.

Procédure pas à pas : création d'un formulaire pour rechercher des données dans une application Windows

Fournit des instructions pas à pas pour créer un Windows Form qui recherche des enregistrements dans une base de données selon des valeurs entrées par un utilisateur.

Procédure pas à pas : création d'un contrôle utilisateur prenant en charge la liaison de données simple

Fournit des instructions pas à pas pour créer un contrôle personnalisé avec des attributs spécifiques à l'exploitation de la liaison de données des Windows Forms.

Procédure pas à pas : affichage de données sur un formulaire dans une application Windows

Fournit des détails pas à pas sur l'interrogation de données provenant d'une base de données et leur affichage sur un Windows Form.

Procédure pas à pas : affichage de données connexes sur un formulaire dans une application Windows

Fournit des instructions pas à pas pour afficher des données issues de deux tables connexes dans un Windows Form.

Rubriques de procédure pas à pas — accès aux données ASP.NET (Visual Studio)

Fournit des liens vers les rubriques de procédure pas à pas qui couvrent l'accès aux données dans les pages Web.

Exemples

Exemples de données

Ces exemples illustrent l'utilisation de données dans des applications Visual Basic.

Voir aussi

Concepts

Comment faire en Visual Basic

Déploiement (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches de déploiement Visual Basic largement utilisées. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Choix d'une stratégie de déploiement

Compare les technologies ClickOnce et Windows Installer.

Déploiement d'applications 64 bits

Décrit les considérations générales relatives au déploiement d'une application 64 bits.

Ajout de composants requis personnalisés

Décrit comment installer des packages pour des composants partagés ou des composants système qui sont requis pour l'exécution de votre application.

Tâches de déploiement de Windows Installer

Cette section contient des tâches de déploiement Windows Installer courantes.

Déploiement avec ClickOnce

Comment : publier une application ClickOnce

Explique comment publier une application ClickOnce sur un serveur Web, un partage de fichiers ou un support amovible.

Comment : spécifier les fichiers publiés via ClickOnce

Explique comment publier ou installer certains fichiers selon des conditions.

Comment : modifier la langue de publication pour une application ClickOnce

Explique comment déterminer la langue et culture en vigueur sur votre ordinateur de développement. Si vous publiez une application localisée, vous devez spécifier une langue et une culture qui correspondent à la version localisée.

Comment : installer les composants requis avec une application ClickOnce

Explique comment choisir un jeu de composants préalables à empaqueter avec votre application.

Comment : gérer des mises à jour pour une application ClickOnce

Décrit la spécification de la manière et du moment de l'exécution des contrôles de mise jour, du caractère obligatoire des mises à jour et de l'emplacement où l'application doit rechercher ces dernières.

Comment : activer les paramètres de sécurité ClickOnce

Explique comment activer des paramètres de sécurité pour publier l'application.

Comment : signer des manifestes d'application et de déploiement

Explique comment affecter un nom fort à un manifeste.

Comment : déployer une application 64 bits à l'aide de ClickOnce

Décrit comment publier une application 64 bits avec ClickOnce.

Publication d'applications

Comment : publier une application ClickOnce

Explique comment publier une application ClickOnce sur un serveur Web, un partage de fichiers ou un support amovible.

Comment : spécifier les fichiers publiés via ClickOnce

Explique comment publier ou installer certains fichiers selon des conditions.

Comment : modifier la langue de publication pour une application ClickOnce

Explique comment déterminer la langue et culture en vigueur sur votre ordinateur de développement. Si vous publiez une application localisée, vous devez spécifier une langue et une culture qui correspondent à la version localisée.

Comment : installer les composants requis avec une application ClickOnce

Explique comment choisir un jeu de composants préalables à empaqueter avec votre application.

Comment : gérer des mises à jour pour une application ClickOnce

Décrit la spécification de la manière et du moment de l'exécution des contrôles de mise jour, du caractère obligatoire des mises à jour et de l'emplacement où l'application doit rechercher ces dernières.

Comment : incrémenter automatiquement la version de publication ClickOnce

Décrit des incréments du numéro de révision de la Version de publication lorsque vous publiez une application.

Comment : spécifier le mode d'installation ClickOnce

Explique comment déterminer si l'application sera disponible hors connexion ou en ligne.

Comment : définir les propriétés d'un projet de déploiement

Décrit des catégories de propriétés de déploiement : générales et dépendantes de la configuration.

Comment : spécifier l'installation par utilisateur ou par ordinateur

Explique comment spécifier si vous souhaitez que l'application soit installée pour tous les utilisateurs de l'ordinateur, ou uniquement pour celui qui procède à son installation.

Signer des manifestes

Comment : signer des manifestes d'application et de déploiement

Explique comment affecter un nom fort à un manifeste.

Comment : temporiser la signature d'un assembly (Visual Studio)

Explique comment utiliser une signature partielle ou temporisée pour fournir la clé publique, en différant l'ajout de la clé privée jusqu'à la mise à disposition de l'assembly.

Déploiement d'applications sécurisées

Comment : définir une zone de sécurité pour une application ClickOnce

Explique comment démarrer avec un jeu d'autorisations de base, puis ajouter une à une les autorisations nécessaires à votre application.

Déploiement avec des projets d'installation (Windows Installer)

Comment : créer ou ajouter un projet d'installation

Explique comment créer des projets d'installation utilisés pour créer des fichiers Windows Installer (.msi). Les projets d'installation sont de deux types : standard et Web.

Procédure pas à pas : déploiement d'une application Windows

Fournit des instructions pour créer un programme d'installation destiné à une application Windows, qui définit des raccourcis et des associations de fichiers, ajoute une entrée au Registre, affiche des boîtes de dialogue personnalisées et vérifie la version de Microsoft Internet Explorer pendant l'installation.

Procédure pas à pas : création d'une action personnalisée

Montre le processus de création d'une action personnalisée DLL permettant de diriger un utilisateur vers une page Web à la fin d'une installation.

Comment : créer un programme d'installation Windows pour une plate-forme 64 bits

Décrit la création de programmes d'installation destinés à des applications et des composants 64 bits.

Comment : installer les composants requis dans le déploiement de Windows Installer

Explique comment choisir un jeu de composants préalables à empaqueter avec votre application.

Autres projets de déploiement

Comment : créer ou ajouter un projet de module de fusion

Crée un fichier de module de fusion (.msm) qui inclut la totalité des fichiers, ressources et entrées du Registre, ainsi que la logique de configuration, de votre composant.

Comment : créer ou ajouter un projet CAB

Crée des fichiers .cab qui peuvent être utilisés pour télécharger des composants sur un navigateur Web. Utilisez cette option lorsque vous voulez que le code s'exécute sur un ordinateur client plutôt que sur un serveur.

Comment : ajouter des modules de fusion à un projet de déploiement

Vous permet de créer vos propres modules de fusion avec Visual Studio, ou d'utiliser des modules de fusion mis à votre disposition pour de nombreux composants standard, par Microsoft ou des vendeurs tiers.

Voir aussi

Concepts

Comment faire en Visual Basic

Applications Smart Device (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches Visual Basic largement utilisées pour des applications Smart Device. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Comment : créer des applications smart device à l'aide de Visual Basic ou Visual C#

Décrit la création de projets managés destinés à des périphériques, qui suivent le même processus général que la création de projets de bureautique.

Procédure pas à pas : création d'applications Windows Forms pour un périphérique

Illustre la création d'une application Windows Forms simple à l'aide de Visual Basic .NET ou Visual C#, puis son exécution sur l'émulateur Pocket PC.

Procédure pas à pas : Hello World : exemple COM Interop pour les smart devices

Affiche la combinaison dans une même solution d'un objet COM simple et d'une application managée.

Comment : modifier l'orientation et la résolution des formulaires (smart devices)

Montre comment modifier des propriétés d'orientation (rotation) par défaut ; s'applique également si vous avez installé un Kit de développement dans lequel ces propriétés sont incorrectes ou manquantes.

Comment : ajouter un objet métier en tant que source de données (smart devices)

Illustre l'ajout d'un objet métier en tant que source de données dans vos projets Smart Device.

Travailler avec des données sur des Smart Devices

Comment : créer des requêtes paramétrées (Smart Devices)

Décrit la création de requêtes paramétrées ; suppose que votre fenêtre Sources de données dispose d'une base de données SQL Server Mobile.

Comment : créer des applications maître/détail (smart devices)

Montre des applications maître/détail ; suppose que votre fenêtre Sources de données dispose d'une base de données SQL Server Mobile dont les tables présentent des relations.

Comment : ajouter un service Web en tant que source de données (smart devices)

Montre comment ajouter un service Web en tant que une source de données dans vos projets Smart Device.

Comment : ajouter une base de données SQL Server en tant que source de données (smart devices)

Montre comment utiliser des bases de données SQL Server en tant que sources de données dans vos projets smart device managés.

Déployer des applications Smart Device

Procédure pas à pas : empaquetage d'une solution smart device pour le déploiement

Montre comment utiliser Visual Studio 2005 pour intégrer votre application et ses ressources à un fichier CAB afin que ce dernier puisse être déployé sur le Smart Device d'un utilisateur final.

Comment : signer une application Visual Basic ou Visual C# (smart devices)

Montre les étapes pour signer une application de périphérique managée.

Utiliser des contrôles

Procédure pas à pas : création de contrôles utilisateur pour les périphériques

Crée une bibliothèque de contrôles et développe un contrôle utilisateur à placer dans celle-ci. Décrit la différence entre un contrôle utilisateur (combinaison de contrôles Windows Forms au sein d'une même entité réutilisable) et un contrôle personnalisé (contrôle qui requiert une interface utilisateur ou des fonctionnalités non disponibles par l'intermédiaire des contrôles standard).

Débogage et test d'applications

Différences entre les débogueurs de projets smart device et les débogueurs de projets bureautiques

Décrit les différences entre les différents débogueurs de périphériques, qui prennent en charge la plupart des fonctionnalités

des débogueurs de bureau, ainsi que les exceptions.

Procédure pas à pas : débogage d'un Windows Form

Illustre le débogage d'un Windows Form.

Procédure pas à pas : débogage d'un formulaire Web

Illustre le débogage d'un formulaire Web.

Procédure pas à pas : débogage d'un service Web XML

Illustre le débogage d'un service Web XML.

Exemples

Exemples et procédures pas à pas (projets smart device)

Ces exemples illustrent les techniques utilisées pour répondre aux défis de la programmation de différents dispositifs.

Voir aussi

Concepts

Comment faire en Visual Basic

Développement Smart Device, comment faire

Programmation Office (Comment faire en Visual Basic)

Cette page contient des liens vers de l'aide sur les tâches de programmation Office largement utilisées. Ils affichent un exemple de code Visual Basic dans le contexte de Visual Studio Tools pour Office. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Comment : mettre à niveau des solutions à partir de Visual Studio Tools pour Office

Décrit les actions à entreprendre pour profiter des nouveaux outils et du nouveau système de projet fournis dans la version la plus récente de Microsoft Visual Studio Tools pour Microsoft Office System.

Comment : installer les assemblys PIA (Primary Interop Assembly) d'Office

Décrit les différentes étapes de l'installation des assemblys PIA (Primary Interop Assembly) requis de Microsoft Office 2003 sur les ordinateurs des utilisateurs finaux.

Comment : configurer les informations de configuration d'une solution Office

Explique comment vous pouvez ajouter manuellement un fichier .config à votre projet Office.

Comment : créer des projets Visual Studio Tools pour Office

Explique comment utiliser l'Assistant Projet Microsoft Office pour créer un projet Word ou Excel à l'aide d'un document ou classeur, nouveau ou déjà existant.

Comment : automatiser d'autres applications Office que Word, Excel ou Outlook

Explique comment utiliser un code managé pour automatiser d'autres applications Office, tel que Microsoft Office PowerPoint 2003, dans Visual Studio.

Comment : ajouter des contrôles à des documents Office

Explique comment ajouter des contrôles Windows Forms à des documents Excel 2003 et Word 2003 au moment du design ou au moment de l'exécution.

Procédure pas à pas : octroi et suppression d'autorisations pour une solution Office

Fournit une démonstration pas à pas des notions de base en matière de sécurité dans les solutions Visual Studio Tools pour Office.

Comment : déployer des solutions Office

Décrit trois façons différentes de déployer une solution Office : déploiement du document et de l'assembly sur un réseau, déploiement local de ces derniers, ou déploiement de l'assembly sur un réseau et déploiement local du document.

Applications Excel

Comment : exécuter des calculs Excel par programme

Montre comment utiliser la méthode Calculate pour exécuter tous les calculs présents dans une plage, ainsi que dans tous les classeurs ouverts.

Comment : gérer les erreurs dans les projets Office

Explique comment l'interaction entre du code managé et du code non managé affecte la gestion des erreurs dans les projets Microsoft Office Word 2003 et Microsoft Office Excel 2003.

Comment : afficher une chaîne dans une cellule de feuille de calcul

Montre comment afficher du texte dans une cellule par programme. Vous pouvez créer une plage nommée pour référencer la cellule.

Applications Word

Comment : utiliser les boîtes de dialogue intégrées dans Word

Fournit un exemple qui indique comment tirer parti des boîtes de dialogue intégrées de Word pour afficher les entrées utilisateur.

Comment : créer des barres d'outils Office par programme

Fournit un exemple montrant comment créer par programme une barre d'outils contenant deux boutons dans Word 2003.

Applications Outlook

Comment : créer un élément de messagerie

Fournit un exemple qui montre comment créer un élément de messagerie électronique dans Microsoft Outlook 2003.

Comment : ajouter des icônes personnalisées aux éléments de menu et de barre d'outils

Fournit un exemple qui applique une icône à une commande dans un menu personnalisé.

Données dans les solutions Office

Comment : mettre en cache par programme une source de données dans un document Office

Indique comment ajouter par programme une source de données à un cache de document en appelant la méthode **StartCaching** et en passant la source de données.

Comment : mapper des schémas à des documents Word dans Visual Studio

Montre comment mapper un schéma XML à un document pendant que celui-ci est ouvert dans un projet Visual Studio Tools pour Office, à l'aide des mêmes outils Word 2003 que ceux que vous utilisez lorsque le document est ouvert en dehors de Visual Studio.

Procédure pas à pas : liaison d'une cellule de feuille de calcul à un champ de base de données

Vous guide pas à pas au sein des notions de base de la liaison d'un champ de données unique dans une base de données SQL Server à une plage nommée dans Excel 2003.

Comment : remplir des documents avec les données d'objets

Montre comment l'accès aux données stockées d'objets de documents Word 2003 est semblable l'utilisation de projets Windows Forms.

Balises actives et volets Actions

Procédure pas à pas : Insertion de texte dans un document à partir d'un volet Actions

Vous guide pas à pas au sein du processus de création d'un volet Actions qui remplit un document Word 2003 avec le texte que l'utilisateur tape dans ce volet.

Comment : créer et modifier des propriétés de document personnalisées

Décrit comment vous pouvez créer et modifier des propriétés de document personnalisées si vous souhaitez stocker des informations supplémentaires avec un document ou un classeur.

Exemples

Exemples de développement Office

Ces exemples montrent comment utiliser Microsoft Visual Studio 2005 Tools pour Microsoft Office System afin de bénéficier des fonctionnalités disponibles dans Microsoft Office Word 2003 et dans Microsoft Office Excel 2003.

Voir aussi

Concepts

Comment faire en Visual Basic

Comment faire dans Visual Studio Tools pour Office

Applications Web (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches Visual Basic largement utilisées pour créer des applications Web. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Nouveautés du développement Web pour Visual Studio

Décrit Visual Web Developer, un jeu d'outils et d'utilitaires qui permet de créer des applications Web ASP.NET.

Procédure pas à pas : création d'une page Web de base dans Visual Web Developer

Fournit une première introduction à la création et la modification d'une page Web ASP.NET dans Visual Studio, notamment une vue d'ensemble de l'environnement de développement Web.

Procédure pas à pas : modification de base du code HTML dans Visual Web Developer

Présente les fonctionnalités d'édition de code HTML de Visual Web Developer.

Procédure pas à pas : débogage des pages Web dans Visual Web Developer

Fournit une vue d'ensemble de l'utilisation du débogueur avec des pages Web.

Utiliser des Web Forms et des contrôles

Comment : valider des entrées requises pour les contrôles serveur ASP.NET

Montre comment spécifier qu'un utilisateur doit fournir des informations dans un contrôle spécifique en ajoutant un contrôle **RequiredFieldValidator** à la page et en le liant au contrôle requis.

Comment : répondre à la sélection d'un utilisateur dans un contrôle serveur Web CheckBox

Fournit des instructions relatives à l'écriture d'un gestionnaire d'événements qui réponde directement lorsque l'utilisateur clique sur un contrôle CheckBox.

Comment : déterminer la sélection dans un contrôle serveur Web List

Donne des instructions pour associer trois valeurs à une sélection de liste : la valeur d'index de la sélection, le texte (valeur d'affichage) de la sélection et la valeur éventuelle de la sélection.

Comment : permettre aux utilisateurs de sélectionner des éléments dans des contrôles serveur Web DataList

Fournit des instructions relatives à la définition d'un bouton Sélectionner qui permet aux utilisateurs d'afficher un élément en surbrillance.

Comment : ajouter des lignes et des cellules de façon dynamique à un contrôle serveur Web Table

Fournit des instructions pour créer des lignes et des cellules d'un contrôle Table au moment de l'exécution.

Comment : ajouter des contrôles à une page Web ASP.NET par programme

Utilise un événement de modification de liste déroulante pour ajouter des étiquettes à une page Web Forms au moment de l'exécution.

Comment : ajouter de manière dynamique un script client à des pages Web ASP.NET

Fournit des informations sur l'utilisation d'un code serveur pour ajouter un script client à une page par programme.

Déploiement d'un site Web

Procédure pas à pas : publication d'un site Web

Fournit une visite guidée de la compilation d'un site Web dans des assemblys qui peuvent être déployés sur un serveur Web.

Procédure pas à pas : copie d'un site Web à l'aide de l'outil Copier le site Web

Fournit une visite guidée de la création d'un site Web et de la copie des fichiers sur un serveur Web de production.

Développer un site Web

Comment : créer des pages de contenu pour une page maître ASP.NET (Visual Studio)

Fournit des instructions pour créer une page de contenu de type Web ASP.NET, associée à une page maître.

Comment : passer des valeurs entre des pages Web ASP.NET

Fournit des informations sur les actions à effectuer si votre application est redirigée (par navigation) d'une page Web ASP.NET à une autre, et si vous souhaitez passer des informations de la page source à la page cible.

Comment : rediriger les utilisateurs vers une autre page

Fournit des instructions pour rediriger des utilisateurs d'une page Web ASP.NET vers une autre page.

Comment : référencer le contenu de la page maître ASP.NET

Montre l'utilisation d'un code dans les pages de contenu pour référencer des propriétés, des méthodes et des contrôles dans la page maître, avec quelques restrictions.

Comment : appliquer des thèmes ASP.NET

Montre comment définir un thème au niveau d'une application et au niveau d'une page.

Types de validation pour les contrôles serveur ASP.NET

Répertorie les types de contrôles de validation disponibles, ainsi que la façon de les utiliser.

Procédure pas à pas : création de pages Web maître/détail dans Visual Studio

Montre plusieurs manières de travailler avec des données dans les contrôles multiples et à partir de tables multiples, notamment avec celles qui affichent une relation maître/détail.

Procédure pas à pas : création d'éléments réutilisables avec des contrôles utilisateur ASP.NET

Montre comment créer un contrôle utilisateur ASP.NET qui agit en tant que contrôle sélecteur ; ce contrôle se compose de deux zones de liste, dont l'une affiche un ensemble de choix (la source).

Procédure pas à pas : création de pages Web pour les périphériques mobiles

Crée deux pages Web héritant de la classe MobilePage qui sont conçues pour un périphérique mobile.

Procédure pas à pas : personnalisation d'un site Web à l'aide de thèmes dans Visual Studio

Illustre comment utiliser des thèmes pour appliquer un aspect homogène aux pages et aux contrôles de votre site Web.

Procédure pas à pas : intégration du traçage ASP.NET avec le traçage System.Diagnostics

Décrit comment intégrer des techniques de suivi ASP.NET avec System.Diagnostics et enregistrer tous les messages de trace dans une sortie de traçage unique.

Procédure pas à pas : utilisation de ressources pour la localisation avec ASP.NET

Montre une manière efficace de créer des pages Web localisées à l'aide de ressources destinées au texte de votre page et de contrôles fondés sur la langue et la culture de l'utilisateur.

Procédure pas à pas : création d'une page WebPart dans Visual Web Developer

Montre les composants et les tâches essentiels pour travailler avec les WebParts dans un concepteur visuel tel que Visual Studio.

Effectuer une mise à niveau de la technologie ASP
Conversion d'un projet Web de Visual Studio .NET

Inclut des informations relatives à différents aspects de la conversion à Visual Studio 2005 de projets .NET Visual Studio 2003.

Dépannage de la conversion vers Visual Web Developer

Décrit des situations et suggère des solutions pour les cas dans lesquelles vos applications Web se comportent différemment à l'issue de leur conversion.

Voir aussi
Concepts
Comment faire en Visual Basic
Comment faire dans Visual Web Developer

Services Web (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches Visual Basic largement utilisées pour créer des applications de service Web. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Procédure pas à pas : création et utilisation d'un service Web ASP.NET dans Visual Web Developer

Illustre la création d'un service Web XML simple dans Visual Web Developer.

Procédure pas à pas : création d'un service Web XML en utilisant Visual Basic ou Visual C#

Explique comment créer un service Web XML qui convertit en degrés Celsius des températures mesurées en degrés Fahrenheit.

Comment : créer des projets de service Web ASP.NET

Fournit des instructions sur la création d'un service Web XML à l'aide du modèle de projet de service Web ASP.NET.

Comment : établir une connexion à des données dans un service Web

Décrit comment créer une connexion entre votre application et les données retournées issues d'un service Web à l'aide de l'**Assistant Configuration de source de données**.

Créer un service Web

Procédure pas à pas : création et utilisation d'un service Web ASP.NET dans Visual Web Developer

Illustre la création d'un service Web XML simple dans Visual Web Developer.

Procédure pas à pas : création d'un service Web XML en utilisant Visual Basic ou Visual C#

Explique comment créer un service Web XML qui convertit en degrés Celsius des températures mesurées en degrés Fahrenheit.

Comment : créer des projets de service Web ASP.NET

Fournit des instructions sur la création d'un service Web XML à l'aide du modèle de projet de service Web ASP.NET.

Comment : établir une connexion à des données dans un service Web

Décrit comment créer une connexion entre votre application et les données retournées issues d'un service Web à l'aide de l'**Assistant Configuration de source de données**.

Déboguer et déployer un service Web

Exécution pas à pas d'un service Web XML

Explique comment basculer entre une application cliente et un service Web XML.

Comment : déployer des services Web XML en code managé

Explique comment déployer un service Web à l'aide de différentes méthodes.

Comment : déboguer des services Web XML en code managé

Explique comment déboguer un service Web à l'aide de différentes méthodes.

Appeler un service Web

Procédure pas à pas : accès à un service Web XML en utilisant Visual Basic ou Visual C#

Décrit le processus permettant d'accéder à un service Web XML à partir d'une application créée avec Visual Basic ou Visual C#.

Comment : effectuer un appel asynchrone à partir d'un client de service Web

Montre comment appeler un service Web de manière asynchrone à partir d'une application cliente.

Comment : explorer les services Web XML existants créés à l'aide d'ASP.NET

Fournit des informations concernant la fonctionnalité du service Web, y compris les méthodes qu'il implémente, leurs paramètres et leurs types de retour.

Comment : appeler un service Web

Illustre l'appel à une méthode de service Web en vue de définir une valeur de chaîne.

Comment : ajouter un service Web XML à un projet Web existant en code managé

Explique comment ajouter un service Web à un projet Web existant.

Voir aussi

Concepts

Comment faire en Visual Basic

Comment faire dans Visual Web Developer

Applications Windows (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches Visual Basic largement utilisées pour créer des applications Windows. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Structure d'un programme Visual Basic

Fournit la description générale d'un programme Visual Basic simple.

Procédure pas à pas : création d'un Windows Form simple

Décrit comment créer une simple application « Hello, World ».

Utiliser des Windows Forms et des contrôles

Nouveautés dans les Windows Forms et les contrôles Windows Forms

Énumère les fonctionnalités nouvelles et améliorées disponibles dans cette version de Visual Studio.

Nouveautés dans les Windows Forms pour les utilisateurs de Visual Basic 6.0

Met en surbrillance les modifications clés qui concernent les utilisateurs Visual Basic 6.

Travailler avec des formulaires

Lors du design

Comment : choisir le formulaire de démarrage dans une application Windows

Explique comment définir le premier formulaire devant s'afficher au lancement de l'application.

Comment : connecter plusieurs événements à un même gestionnaire d'événements dans les Windows Forms

Explique comment assigner la même fonctionnalité à plusieurs contrôles par le biais d'événements.

Comment : créer une interface utilisateur à plusieurs volets à l'aide des Windows Forms

Explique comment créer une interface utilisateur à plusieurs volets semblable à celle de Microsoft Outlook.

Comment : ajouter des images d'arrière-plan à des Windows Forms

Explique comment placer une image d'arrière-plan dans un contrôle ou sur le formulaire proprement dit.

Comment : définir des info-bulles pour les contrôles d'un Windows Form au moment du design

Explique comment définir des composants Tooltip dans le code ou dans le concepteur.

Comment : réassignation de contrôles existants à un parent différent

Donne des instructions pour affecter des contrôles existants à un nouveau conteneur parent.

Comment : ajouter des contrôles ActiveX aux Windows Forms

Explique comment manipuler des contrôles ActiveX hérités (legacy).

Comment : fournir de l'aide dans une application Windows

Explique comment utiliser le composant **HelpProvider** pour lier des contrôles à des fichiers d'aide.

Comment : créer des touches d'accès rapide pour des contrôles Windows Forms

Fournit des informations sur la création de raccourcis clavier prédéfinis.

Comment : hériter des Windows Forms

Explique comment créer des formulaires hérités dans le code.

Comment : hériter de contrôles Windows Forms existants

Explique comment créer un contrôle hérité.

Lors de l'exécution

Comment : ajouter des contrôles dans une collection au moment de l'exécution, ou comment les supprimer

Explique comment ajouter et supprimer des contrôles sur un panneau au moment de l'exécution.

Comment : activer des styles visuels de Windows XP

Donne des instructions pour utiliser la propriété **EnableVisualStyles** pour donner à des formulaires et des contrôles l'aspect standard de Windows XP.

Comment : définir l'invisibilité d'un formulaire de démarrage Windows

Fournit des instructions sur la définition de la visibilité initiale d'un formulaire au moment de l'exécution.

Comment : conserver un Windows Form au premier plan

Explique comment faire en sorte qu'un Windows Form reste devant tous les autres formulaires au moment de l'exécution.

Comment : afficher des Windows Forms modaux et non modaux

Explique comment afficher une boîte de dialogue sous une forme modale ou non modale.

Comment : rendre votre contrôle invisible au moment de l'exécution

Donne des instructions pour créer un contrôle utilisateur invisible au moment de l'exécution.

Contrôles communs

Contrôles TextBox

Comment : sélectionner du texte dans le contrôle TextBox Windows Forms

Explique comment mettre du texte en surbrillance dans une zone de texte.

Comment : insérer des guillemets dans une chaîne (Windows Forms)

Explique comment ajouter des guillemets à une chaîne dans une zone de texte.

Comment : afficher les barres de défilement dans le contrôle RichTextBox Windows Forms

Décrit les nombreuses options disponibles pour les barres de défilement dans le contrôle RichTextBox.

Comment : créer une zone de texte en lecture seule (Windows Forms)

Explique comment empêcher que le contenu d'une zone de texte ne soit modifié.

Comment : créer une zone de texte pour mot de passe avec le contrôle TextBox Windows Forms

Explique comment masquer la valeur tapée dans une zone de texte.

Comment : contrôler le point d'insertion dans un contrôle TextBox Windows Forms

Explique comment spécifier la position à laquelle le point d'insertion doit apparaître lorsque le focus arrive pour la première fois dans un contrôle d'édition.

Comment : récupérer des données du Presse-papiers

Explique comment accéder aux données stockées dans le Presse-papiers.

Comment : ajouter des données au Presse-papiers

Décrit une méthode utilisée pour insérer par programme des informations dans le Presse-papiers.

Comment : lier des données au contrôle MaskedTextBox

Montre comment reformater les données lorsque celles de la base de données ne correspondent pas au format attendu par votre définition de masque.

Procédure pas à pas : utilisation du contrôle MaskedTextBox

Montre les fonctionnalités clés du contrôle MaskedTextBox.

Contrôles RichTextBox

Comment : charger des fichiers dans le contrôle RichTextBox Windows Forms

Explique comment charger un fichier existant dans le contrôle **RichTextBox**.

Comment : afficher les barres de défilement dans le contrôle RichTextBox Windows Forms

Décrit les nombreuses options disponibles pour les barres de défilement dans le contrôle **RichTextBox**.

Comment : définir les attributs de police du contrôle RichTextBox Windows Forms

Explique comment définir la famille de polices, ainsi que la taille, le style et la couleur du texte dans le contrôle **RichTextBox**.

Comment : définir les retraits, les retraits négatifs de première ligne et les listes à puces avec le contrôle RichTextBox Windows Forms

Explique comment mettre en forme des paragraphes dans le contrôle **RichTextBox**.

Comment : activer les opérations glisser-déplacer avec le contrôle RichTextBox Windows Forms

Explique comment faire glisser des données dans le contrôle **RichTextBox**.

Comment : afficher des liens de style Web avec le contrôle RichTextBox Windows Forms

Explique comment créer un lien vers des sites Web à partir du contrôle **RichTextBox**.

Contrôles Button

Comment : répondre à un clic du contrôle Button Windows Forms

Présente l'utilisation la plus élémentaire d'un bouton dans un formulaire d'une application Windows.

Comment : désigner un bouton Windows Forms comme bouton Accepter à l'aide du concepteur

Montre comment installer un bouton comme le bouton d'acceptation.

Comment : désigner un bouton Windows Forms comme bouton Annuler à l'aide du concepteur

Montre comment installer un bouton comme le bouton d'annulation.

Contrôles CheckBox et RadioButton

Contrôle CheckBox

Comment : définir des options avec les contrôles CheckBox Windows Forms

Explique comment utiliser une case à cocher pour définir des options telles que les propriétés d'un objet.

Comment : répondre à un clic du contrôle CheckBox Windows Forms

Explique comment utiliser une case à cocher pour déterminer les actions effectuées par votre application.

Contrôles RadioButton

Comment : grouper des contrôles RadioButton Windows Forms en un ensemble fonctionnant indépendamment

Explique comment regrouper en ensembles des cases d'option, parmi lesquelles une seule peut être activée.

Contrôles ListBox, ComboBox et CheckedListBox

Comment : lier un contrôle ComboBox ou ListBox Windows Forms aux données

Explique comment lier un contrôle de liste à une source de données.

Comment : créer une table de correspondance pour un contrôle ComboBox, ListBox ou CheckedListBox Windows Forms

Explique comment afficher et stocker les données d'un formulaire dans des formats utiles.

Comment : ajouter et supprimer des éléments d'un contrôle ComboBox, ListBox ou CheckedListBox Windows Forms

Explique comment ajouter ou supprimer des éléments d'une liste d'éléments d'un contrôle.

Comment : accéder à des éléments spécifiques d'un contrôle ComboBox, ListBox ou CheckedListBox Windows Forms

Explique comment déterminer par programme quel élément d'une liste apparaît à une position donnée.

Comment : trier le contenu d'un contrôle ComboBox, CheckedListBox ou ListBox Windows Forms

Explique comment trier les données d'une liste au niveau de sa source.

Contrôle CheckedListBox

Comment : déterminer des éléments cochés dans le contrôle CheckedListBox Windows Forms.

Explique comment déterminer, dans une liste, quels éléments sont sélectionnés.

Contrôle DataGridView

Comment : lier des données au contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser l'option **Choisir une source de données** sur la balise active du contrôle pour établir une connexion aux données.

Comment : créer un formulaire maître/détail utilisant deux contrôles DataGridView Windows Form

Fournit un code qui illustre comment afficher des données issues de deux tables de base de données connexes, afin que les valeurs affichées dans un contrôle DataGridView spécifique dépendent de la ligne en cours de sélection dans un autre contrôle.

Comment : valider des données dans le contrôle DataGridView Windows Forms

Fournit un code qui montre comment valider une entrée d'utilisateur en vue d'empêcher d'éventuelles erreurs de formatage dans la saisie des données.

Comment : gérer les erreurs qui se produisent lors de l'entrée de données dans le contrôle DataGridView Windows Forms

Fournit un code qui illustre comment gérer des erreurs de saisie de données issues de la source de données lorsque l'utilisateur essaie de valider une nouvelle valeur.

Comment : empêcher l'ajout et la suppression de lignes dans le contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser la balise active du contrôle pour empêcher des utilisateurs d'ajouter ou de supprimer des lignes.

Comment : spécifier des valeurs par défaut pour les nouvelles lignes dans le contrôle DataGridView Windows Forms

Explique comment effectuer un remplissage préalable de la ligne destinée aux nouveaux enregistrements pour consigner l'heure de saisie des données.

Comment : créer un contrôle DataGridView Windows Forms indépendant

Fournit un code qui illustre comment remplir manuellement le contrôle avec des données.

Comment : ajouter une colonne indépendante à un contrôle DataGridView Windows Forms lié aux données

Explique comment compléter des données issues d'une source de données liée en affichant des colonnes supplémentaires non liées.

Comment : afficher des images dans les cellules du contrôle DataGridView Windows Forms

Explique comment créer une colonne d'image dont chaque cellule affiche une icône.

Comment : héberger des contrôles dans des cellules DataGridView Windows Forms

Explique comment implémenter l'interface IDataGridViewEditingControl et créer des types personnalisés dérivés de DataGridViewCell et de DataGridViewColumn pour afficher un contrôle DateTimePicker lorsqu'une cellule est en mode édition.

Procédure pas à pas : validation des données dans le contrôle DataGridView Windows Forms

Explique comment valider une entrée d'utilisateur en vue d'empêcher d'éventuelles erreurs de formatage dans la saisie des données.

Procédure pas à pas : gestion des erreurs qui se produisent lors de la saisie de données dans le contrôle DataGridView Windows Forms

Explique comment gérer des erreurs de saisie de données issues de la source de données lorsque l'utilisateur essaie de valider une nouvelle valeur.

Procédure pas à pas : création d'un contrôle DataGridView Windows Forms non lié

Montre comment remplir manuellement le contrôle avec des données.

Présentation et formatage du contrôle DataGridView

Comment : figer les colonnes du contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser la boîte de dialogue **Modifier les colonnes** pour empêcher le défilement de colonnes spécifiques.

Comment : définir une colonne en lecture seule dans le contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser la boîte de dialogue **Modifier les colonnes** pour empêcher des utilisateurs de modifier des valeurs dans des colonnes spécifiques.

Comment : activer la réorganisation des colonnes dans le contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser la balise active du contrôle pour permettre aux utilisateurs de réordonner des colonnes.

Comment : modifier l'ordre des colonnes dans le contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser la boîte de dialogue **Modifier les colonnes** pour réordonner des colonnes.

Comment : ajouter et supprimer des colonnes dans le contrôle DataGridView Windows Forms à l'aide du concepteur

Explique comment utiliser les boîtes de dialogue **Ajouter des colonnes** et **Modifier les colonnes** pour remplir et modifier la collection de colonnes.

Contrôles ListView et TreeView

Procédure pas à pas : création d'une interface de style Explorateur avec les contrôles ListView et TreeView à l'aide du concepteur

Illustre la création d'une application Windows d'aspect professionnel qui utilise deux contrôles communs.

Contrôle ListView

Comment : ajouter et supprimer des éléments avec le contrôle ListView Windows Forms

Explique comment ajouter ou supprimer des éléments d'une vue Liste.

Comment : ajouter des fonctions de recherche à un contrôle ListView

Décrit deux manière de fournir des fonctions de recherche à un utilisateur : par correspondance de texte et par recherche d'emplacement.

Comment : sélectionner un élément dans le contrôle ListView Windows Forms

Montre comment sélectionner un élément par programme dans un contrôle ListView Windows Forms.

Comment : afficher des icônes pour le contrôle ListView Windows Forms

Explique comment associer une vue Liste à une liste d'images appropriée pour afficher de grandes ou de petites icônes.

Comment : afficher des sous-éléments en colonnes avec le contrôle ListView Windows Forms

Explique comment afficher en colonnes des informations sur chaque élément d'une liste.

Contrôle TreeView

Comment : définir des icônes pour le contrôle TreeView Windows Forms

Explique comment afficher des icônes pour les nœuds d'une arborescence.

Comment : ajouter et supprimer des nœuds avec le contrôle TreeView Windows Forms

Explique comment ajouter et supprimer des nœuds dans une arborescence.

Comment : identifier le nœud de TreeView sur lequel un clic est effectué (Windows Forms)

Explique comment déterminer quel nœud d'une arborescence a été sélectionné par un clic de souris de sorte que l'application puisse offrir une réponse appropriée.

Contrôles conteneurs

Comment : fractionner une fenêtre horizontalement

Explique comment contrôler l'orientation du séparateur dans le contrôle SplitContainer.

Comment : créer une interface utilisateur à plusieurs volets à l'aide des Windows Forms

Explique comment créer une interface utilisateur à plusieurs volets semblable à celle de Microsoft Outlook.

Comment : étendre des lignes et des colonnes dans un contrôle TableLayoutPanel

Explique comment étendre un contrôle à plusieurs lignes et colonnes adjacentes dans un TableLayoutPanel.

Procédure pas à pas : organisation des contrôles dans les Windows Forms à l'aide d'un TableLayoutPanel

Explique comment créer un formulaire dont la disposition se réorganise de manière appropriée lorsque le formulaire est redimensionné ou lorsque la taille de son contenu change.

Procédure pas à pas : organisation des contrôles dans les Windows Forms à l'aide d'un FlowLayoutPanel

Explique comment créer un formulaire dont la disposition se réorganise de manière appropriée lorsque le formulaire est redimensionné ou lorsque la taille de son contenu change.

Contrôles Image et Picture

Comment : charger une image à l'aide du concepteur (Windows Forms)

Explique comment charger et afficher une image sur un formulaire au moment du design en définissant des propriétés.

Comment : définir des images au moment de l'exécution (Windows Forms)

Explique comment afficher et effacer une image au moment de l'exécution.

Comment : modifier la taille ou l'emplacement d'une image au moment de l'exécution (Windows Forms)

Explique le rôle de la propriété SizeMode et explique comment la définir.

Comment : rogner et mettre des images à l'échelle

Explique comment rogner et mettre à l'échelle par programme des images tramées et vectorielles.

Contrôles de définition de date

Contrôle DateTimePicker

Comment : définir et retourner des dates à l'aide du contrôle DateTimePicker Windows Forms

Décrit, étape par étape, comment définir la date dans le contrôle et accéder à la date sélectionnée par l'utilisateur.

Comment : afficher une date dans un format personnalisé à l'aide du contrôle DateTimePicker Windows Forms

Explique comment utiliser des chaînes de format pour afficher des dates dans le format de votre choix.

Contrôle MonthCalendar

Comment : sélectionner une plage de dates dans le contrôle MonthCalendar Windows Forms

Explique comment sélectionner une plage de dates par programme à partir du contrôle MonthCalendar.

Comment : afficher en gras certains jours à l'aide du contrôle MonthCalendar Windows Forms

Explique comment définir certaines dates à afficher en gras.

Comment : afficher plusieurs mois dans le contrôle MonthCalendar Windows Forms

Explique comment configurer le contrôle **MonthCalendar** de sorte qu'il affiche plusieurs mois simultanément.

Comment : modifier l'apparence du contrôle MonthCalendar Windows Forms

Explique comment personnaliser l'aspect du contrôle **MonthCalendar**.

Accès aux données (pour les Windows Forms)

Nouveautés des données

Fournit des informations et des liens vers de nouvelles fonctionnalités de données destinées aux applications clientes et de couches de données.

Comment : afficher des données dans un contrôle DataGridView Windows Forms

Explique comment présenter des données dans un contrôle DataGridView.

Procédure pas à pas : passage de données entre formulaires dans une application Windows

Fournit des instructions pas à pas pour passer des valeurs d'un formulaire à un autre dans une application.

Procédure pas à pas : affichage de données sur un formulaire dans une application Windows

Fournit des instructions pas à pas pour interroger des données issues d'une base de données, puis les afficher dans un formulaire.

Procédure pas à pas : création d'un contrôle utilisateur prenant en charge la liaison de données simple

Fournit des instructions pas à pas pour créer un contrôle personnalisé avec des attributs spécifiques à l'exploitation de la liaison de données des Windows Forms.

Procédure pas à pas : création d'un formulaire pour rechercher des données dans une application Windows

Fournit des instructions pas à pas pour créer un formulaire qui recherche des enregistrements dans une base de données selon des valeurs entrées par un utilisateur.

Procédure pas à pas : connexion aux données dans des objets

Fournit des instructions pas à pas pour créer une connexion entre votre application et les propriétés d'un objet.

Menus et barres d'outils

Contrôle ToolStrip

Comment : créer un ToolStrip Windows Forms de base avec des éléments standard à l'aide du concepteur

Montre comment créer un ToolStrip et ajouter sept contrôles ToolStripButton qui représentent des tâches typiques.

Comment : positionner un ToolStripItem sur un ToolStrip

Explique comment positionner un **ToolStripButton** à l'extrême gauche ou droite d'un **ToolStrip**.

Comment : désactiver des ToolStripMenuItems à l'aide du concepteur

Explique comment désactiver à la fois des menus entiers et des commandes de menu individuelles.

Comment : masquer des ToolStripMenuItems à l'aide du concepteur

Explique comment masquer à la fois des menus entiers et des commandes de menu individuelles.

Comment : déplacer des ToolStripMenuItems

déplacez l'intégralité de menus du niveau supérieur et leurs éléments de menu vers un autre emplacement sur le MenuStrip.

Comment : modifier l'apparence du texte et des images de ToolStrip dans les Windows Forms

Explique comment définir et modifier la disposition d'un texte et d'images sur des contrôles ToolStripItem.

Menus contextuels

Comment : associer un menu contextuel à un composant NotifyIcon Windows Forms

Indique comment ajouter un menu contextuel à un composant NotifyIcon de sorte qu'il offre un menu de commandes lorsqu'un utilisateur clique sur ce composant avec le bouton droit.

Comment : ajouter et supprimer des éléments de menu avec le composant ContextMenu Windows Forms

Explique comment ajouter et supprimer des éléments du menu contextuel.

Liaison de données avec des contrôles

Comment : gérer des erreurs et des exceptions qui se produisent avec Databinding

Montre l'utilisation du composant BindingSource pour gérer harmonieusement les erreurs qui se produisent lors d'une opération de liaison de données.

Contrôle BindingSource

Comment : lier des contrôles Windows Forms au composant BindingSource à l'aide du concepteur

Explique comment lier des contrôles à une source de données, afin qu'au moment de l'exécution, les utilisateurs puissent modifier et enregistrer des données en rapport avec votre application.

Comment : créer une table de correspondance avec le composant BindingSource Windows Forms

Montre comment utiliser une zone de liste déroulante pour afficher le champ doté de la relation de clé étrangère du parent à la table enfant.

Comment : répercuter des mises à jour de source de données dans un contrôle Windows Forms avec le BindingSource

Montre comment réagir aux modifications intervenues dans la source de données à l'aide du composant **BindingSource**.

Comment : trier et filtrer des données ADO.NET avec le composant BindingSource Windows Forms

Montre l'utilisation du composant **BindingSource** pour appliquer des tris et des filtres aux données affichées.

Comment : établir une liaison à un service Web à l'aide du BindingSource Windows Forms

Montre comment utiliser le composant **BindingSource** pour lier un service Web.

Contrôle BindingNavigator

Comment : naviguer parmi les données avec le contrôle BindingNavigator Windows Forms

Explique comment lier un contrôle BindingNavigator à une source de données.

Comment : parcourir un DataSet avec le contrôle BindingNavigator Windows Forms

Montre l'utilisation d'un contrôle **BindingNavigator** pour parcourir des enregistrements dans un DataSet.

Comment : ajouter des boutons Charger, Enregistrer et Annuler au contrôle BindingNavigator Windows Forms

Lie un contrôle de zone de texte à des données. Ensuite, un contrôle **ToolStrip** ajouté au formulaire est modifié pour intégrer des boutons de chargement, d'enregistrement et d'annulation.

Impression

Comment : créer des travaux d'impression Windows Forms standard

Explique comment utiliser le composant PrintDocument pour imprimer à partir d'un formulaire.

Comment : terminer des travaux d'impression Windows Forms

Explique comment prévenir les utilisateurs qu'un travail d'impression est terminé.

Comment : imprimer un fichier texte composé de plusieurs pages dans les Windows Forms

Décrit l'envoi de texte à l'imprimante.

Comment : afficher le composant PrintDialog

Explique comment afficher la boîte de dialogue et où enregistrer ses propriétés.

Comment : afficher l'aperçu avant impression dans les applications Windows Forms

Décrit la manière dont le contrôle PrintPreviewDialog se présente aux utilisateurs.

Comment : choisir les imprimantes connectées à l'ordinateur d'un utilisateur dans Windows Forms

Décrit la modification de l'imprimante chargée de l'impression à l'aide du composant PrintDialog au moment de l'exécution.

Comment : capturer une entrée d'utilisateur à partir d'un composant PrintDialog au moment de l'exécution

Explique comment utiliser le composant **PrintDialog** pour modifier par programme les options d'impression sélectionnées.

Contrôles utilisateur et contrôles personnalisés

Procédure pas à pas : création d'un contrôle composite à l'aide de Visual Basic

Crée un contrôle utilisateur simple, puis en étend les fonctionnalités par le biais de l'héritage.

Comment : tester le comportement d'un UserControl au moment de l'exécution

Illustre l'utilisation du conteneur de test UserControl fourni par Visual Studio.

Comment : hériter de la classe UserControl

Illustre l'héritage avec la classe **UserControl**.

Comment : créer des contrôles composites

Montre la création un contrôle utilisateur et d'une bibliothèque de classes de contrôles, et l'héritage d'un contrôle utilisateur.

Ajout de contrôles à votre contrôle utilisateur

Montre les éléments fondamentaux du travail avec un contrôle utilisateur.

Ajout de code à votre contrôle utilisateur

Montre les éléments fondamentaux du travail avec un contrôle utilisateur.

Interfaces MDI (MDI, Multiple Document Interface)

Comment : créer des formulaires MDI parents

Explique comment créer le conteneur de plusieurs documents dans une application MDI.

Comment : créer des formulaires MDI enfants

Explique comment créer une ou plusieurs fenêtres opérant dans un formulaire MDI parent.

Comment : réorganiser des formulaires MDI enfants

Explique comment afficher les fenêtres enfants d'une application MDI en mosaïque ou en cascade, et comment les réorganiser.

Comment : déterminer l'enfant MDI actif

Explique comment vérifier la fenêtre enfant qui a le focus et envoyer son contenu au Presse-papiers.

Comment : envoyer des données à l'enfant MDI actif

Explique comment transmettre des informations à la fenêtre enfant active.

Comment : créer une liste des fenêtres MDI avec MenuStrip (Windows Forms)

Explique comment utiliser l'interface MDI pour créer la liste de tous les formulaires enfants actifs dans le menu Fenêtre du parent.

Graphiques

Comment : dessiner une forme avec contour

Décrit comment dessiner une forme.

Comment : créer un dégradé linéaire

Indique comment créer un dégradé linéaire à l'aide de la classe LinearGradientBrush.

Comment : créer un dégradé de tracé

Décrit comment créer un dégradé de tracé à l'aide de la classe PathGradientBrush.

Comment : créer des figures à partir de lignes, de courbes et de formes

Indique comment utiliser un GraphicsPath pour créer des illustrations.

Comment : créer des objets graphiques pour le dessin

Indique comment créer un objet Graphics pour le dessin.

Comment : créer des images miniatures

Décrit comment créer des images miniatures.

Comment : créer du texte vertical

Décrit comment dessiner du texte aligné verticalement avec GDI+.

Comment : aligner le texte écrit

Montre comment mettre en forme du texte GDI+.

Comment : dessiner une ligne dans un Windows Form

Montre comment tracer une ligne.

Comment : faire pivoter, refléter et incliner des images

Décrit comment dessiner des images en leur faisant subir une rotation, une réflexion ou une inclinaison.

Comment : dessiner du texte dans un Windows Form

Décrit comment dessiner du texte.

Comment : charger et afficher des bitmaps

Décrit comment charger et dessiner des bitmaps.

Comment : charger et afficher des métafichiers

Montre comment charger et dessiner des métafichiers.

Comment : rogner et mettre des images à l'échelle

Explique comment rogner et mettre à l'échelle des images tramées et vectorielles.

Localisation et globalisation des Windows Forms

Procédure pas à pas : localisation de Windows Forms

Illustre la création et l'utilisation d'un fichier de ressources pour un projet d'application Windows.

Procédure pas à pas : création d'une présentation qui ajuste la proportion pour la localisation

Montre comment créer une mise en forme qui s'ajuste proportionnellement lorsque vous traduisez les valeurs de chaîne affichées dans d'autres langues.

Comment : prendre en charge la localisation dans les Windows Forms à l'aide du redimensionnement automatique et du contrôle TableLayoutPanel

Montre comment créer des mises en formes qui s'ajustent à différentes longueurs de chaîne de texte que vous ne pouvez pas anticiper au moment de la conception.

Comment : définir la culture et la culture de l'interface utilisateur pour la globalisation des Windows Forms

Explique comment définir les propriétés **CurrentCulture** et **CurrentUICulture** pour déterminer les ressources chargées pour une application et la manière dont les informations sont mises en forme.

Comment : afficher du texte de droite à gauche dans les Windows Forms pour la globalisation

Fournit des instructions sur l'affichage de texte de droite à gauche dans les Windows Forms.

Comment : créer une présentation Windows Forms qui répond bien à la localisation

Montre l'utilisation d'un contrôle **TableLayoutPanel** pour générer un formulaire qui réagit bien à la localisation.

Gérer les ressources de l'application

Comment : récupérer des ressources de type chaîne dans Visual Basic

Montre comment accéder à une ressource de type chaîne à l'aide de l'objet **My.Resources** et la récupérer.

Comment : récupérer une image du Presse-papiers en Visual Basic

Explique comment récupérer une image du Presse-papiers.

Comment : ajouter ou supprimer des ressources

Montre comment utiliser le Concepteur de ressources afin d'ajouter ou de supprimer des ressources pour votre projet.

Comment : ajouter ou supprimer des ressources chaîne

Montre comment utiliser le volet **Chaînes** du Concepteur de ressources afin d'ajouter ou de supprimer des ressources de type chaîne pour votre projet.

Travailler avec les fichiers, les dossiers et les lecteurs

Comment : créer un fichier dans Visual Basic

Montre comment créer un fichier.

Comment : déplacer un fichier dans Visual Basic

Montre comment déplacer un fichier dans un autre répertoire.

Comment : renommer un fichier dans Visual Basic

Montre comment renommer un fichier.

Comment : supprimer un fichier dans Visual Basic

Montre comment supprimer un fichier.

Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic

Montre comment créer une copie d'un fichier dans le même répertoire.

Lire dans des fichiers

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

Montre comment utiliser un StreamReader pour lire un fichier.

Comment : lire des fichiers texte dans Visual Basic

Montre comment lire un fichier texte.

Comment : lire des fichiers texte existants dans Mes documents (Visual Basic)

Montre comment lire un fichier texte dans le répertoire **Mes documents**.

Comment : lire des fichiers binaires dans Visual Basic

Montre comment lire un fichier texte binaire.

Travailler avec des fichiers

Comment : déterminer l'extension d'un fichier dans Visual Basic

Illustre comment déterminer l'extension d'un fichier.

Comment : déterminer l'heure de création d'un fichier dans Visual Basic

Illustre comment déterminer la date de création d'un fichier.

Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic

Illustre comment déterminer le chemin d'accès absolu d'un fichier.

Comment : déterminer si un fichier est masqué dans Visual Basic

Illustre comment déterminer si un fichier est masqué.

Comment : déterminer si un fichier existe dans Visual Basic

Illustre comment déterminer si un fichier existe.

Comment : analyser des chemins d'accès dans Visual Basic

Montre comment utiliser les méthodes **My** pour combiner des chemins d'accès de fichier.

Comment : valider des noms de fichiers et des chemins d'accès dans Visual Basic

Montre comment déterminer si une chaîne représente un nom ou un chemin d'accès de fichier.

Travailler avec des lecteurs

Comment : déterminer le nom de volume d'un lecteur dans Visual Basic

Illustre comment déterminer le nom de volume d'un lecteur.

Comment : déterminer le type d'un lecteur dans Visual Basic

Illustre comment déterminer le type d'un lecteur.

Comment : déterminer l'espace total d'un lecteur dans Visual Basic

Illustre comment déterminer l'espace total d'un lecteur.

Comment : déterminer le répertoire racine d'un lecteur dans Visual Basic

Illustre comment déterminer le répertoire racine d'un lecteur.

Écrire dans des fichiers

Comment : écrire dans des fichiers binaires dans Visual Basic

Montre comment écrire dans un fichier binaire.

Comment : insérer du texte dans les fichiers du répertoire Mes Documents dans Visual Basic

Montre comment créer et écrire dans un nouveau fichier texte du répertoire **Mes documents**.

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Montre comment écrire dans un fichier à l'aide d'un objet StreamWriter.

Travailler avec des Dossiers

Comment : créer un répertoire dans Visual Basic

Montre comment créer un répertoire.

Comment : déterminer les attributs d'un répertoire dans Visual Basic

Illustre comment déterminer les attributs d'un répertoire.

Comment : déplacer un répertoire dans Visual Basic

Montre comment déplacer un répertoire.

Comment : déterminer si un répertoire existe dans Visual Basic

Illustre comment déterminer si un répertoire existe.

Comment : supprimer un répertoire dans Visual Basic

Montre comment supprimer un répertoire.

Journal et événements de traçage

Procédure pas à pas : filtrage de la sortie de My.Application.Log

Décrit comment filtrer la sortie des journaux **My.Application.Log**.

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Décrit comment déterminer l'endroit auquel **My.Application.Log** écrit les informations.

Procédure pas à pas : création d'écouteurs de journalisation personnalisés

Décrit comment créer un écouteur de journalisation personnalisé pour **My.Application.Log**.

Comment : écrire des messages de journal

Décrit comment écrire les informations sur les événements dans les journaux de l'application.

Comment : écrire dans le journal des événements de l'application

Décrit comment configurer **My.Application.Log** pour écrire les informations dans un journal des événements.

Comment : enregistrer des exceptions dans Visual Basic

Décrit comment écrire les informations sur les exceptions dans les journaux de l'application.

Utiliser des modèles Application

Comment : créer un projet d'application Windows

Montre comment créer un projet d'application Windows à l'aide de l'environnement de développement intégré (IDE).

Interagir avec COM

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Décrit l'utilisation d'objets COM existants en tant que base des nouveaux objets.

Procédure pas à pas : création d'objets COM avec Visual Basic 2005

Vous guide pas à pas dans le processus de création d'objets COM avec et sans le modèle de classe COM.

Comment : référencer les objets COM à partir de Visual Basic

Décrit la procédure à suivre pour ajouter des références aux objets COM qui possèdent des bibliothèques de types.

Comment : référencer l'instance en cours d'un objet

Montre comment utiliser le mot clé **Me** pour faire référence à l'instance dans laquelle le code s'exécute actuellement.

Gérer les paramètres de l'application

Comment : lire des paramètres d'application dans Visual Basic

Montre comment accéder aux formulaires d'une application pour déterminer les valeurs des paramètres utilisateur.

Comment : rendre persistants les paramètres utilisateur dans Visual Basic

Montre comment accéder aux formulaires d'une application pour enregistrer les valeurs des paramètres utilisateur mises à jour.

Comment : ajouter ou supprimer des paramètres d'application

Montre comment utiliser la page **Paramètres** du Concepteur de projets pour ajouter ou supprimer des paramètres d'application.

Accéder aux ressources de l'ordinateur

Comment : recevoir des chaînes des ports série dans Visual Basic

Explique comment recevoir une chaîne du port série d'un ordinateur.

Comment : afficher les ports série disponibles dans Visual Basic

Explique comment afficher les ports série disponibles.

Comment : démarrer une application et envoyer des séquences de touches (Visual Basic)

Montre comment démarrer une application et lui envoyer des séquences de touches.

Utiliser le presse-papiers

Comment : écrire dans le Presse-papiers en Visual Basic

Explique comment écrire des données dans le Presse-papiers.

Comment : récupérer une image du Presse-papiers en Visual Basic

Explique comment récupérer une image du Presse-papiers.

Comment : lire le Presse-papiers en Visual Basic

Explique comment lire des données contenues dans le Presse-papiers.

Lire des sons

Comment : récupérer des ressources audio dans Visual Basic

Montre comment utiliser l'objet **My.Resources** pour récupérer une ressource audio.

Comment : lire des sons système dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour lire un son système.

Comment : lire des sons dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour lire en arrière-plan un son d'un fichier audio ou d'une ressource d'application.

Comment : lire des sons en boucle dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour lire un son d'un fichier audio ou d'une ressource d'application en tant que boucle continue.

Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour arrêter un son lu en arrière-plan.

Travailler avec le Registre

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Montre comment utiliser l'objet **My.Computer.Registry** pour créer une clé de Registre et définir ses valeurs.

Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour définir une valeur dans une clé de Registre.

Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour lire une valeur dans une clé de Registre.

Comment : supprimer une clé de Registre dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour supprimer une clé de Registre.

Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic

Montre comment utiliser l'objet **My.Computer.Audio** pour déterminer si une valeur existe dans une clé de Registre spécifique.

Utiliser le journal des événements

Procédure pas à pas : exploration des journaux des événements, des sources d'événements et des entrées

Aborde les principaux domaines de fonctionnalités de l'enregistrement des événements dans votre application Visual Studio.

Comment : effacer les journaux des événements

Explique comment supprimer les entrées consignées dans un journal des événements.

Comment : créer et supprimer des journaux des événements personnalisés

Explique comment créer des journaux des événements personnalisés sur un ordinateur local.

Comment : supprimer les journaux des événements

Explique comment supprimer un journal des événements local, ainsi que toutes les entrées qu'il contient.

Comment : lire les entrées d'un journal des événements

Explique comment lire le contenu des journaux des événements.

Comment : vérifier l'existence d'une source d'événements

Explique comment effectuer une requête pour vérifier si un journal des événements spécifique existe.

Travailler avec des réseaux

Comment : transférer un fichier dans Visual Basic

Montre comment télécharger un fichier et le stocker dans un emplacement distant à l'aide de **My.Computer.Network**.

Comment : vérifier l'état de la connexion dans Visual Basic

Indique comment déterminer si la connexion réseau de l'ordinateur est active.

Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Indique comment utiliser la méthode **Ping** pour déterminer si un ordinateur distant ou hôte est disponible.

Utiliser un contrôle de code source

Comment : ouvrir un projet ou une solution à partir du contrôle de code source

Décrit l'ouverture d'un projet ou d'une solution depuis un contrôle de code source.

Comment : ajouter un projet au contrôle de code source

Fournit des procédures pour l'ajout d'un projet au contrôle de code source.

Écrire des applications sécurisées

Procédure pas à pas : validation de la complexité des mots de passe (Visual Basic).

Montre comment déterminer si une chaîne possède les caractéristiques d'un mot de passe fort.

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Montre comment implémenter une authentification et une autorisation personnalisées et comment substituer l'identité par défaut du thread de l'application.

Procédure pas à pas : chiffrement et déchiffrement de chaînes en Visual Basic

Montre comment chiffrer et déchiffrer des chaînes à l'aide de l'algorithme DES (Data Encryption Standard).

Comment : déterminer si un utilisateur est dans un groupe

Montre comment utiliser l'objet **My.User** pour déterminer le rôle de l'utilisateur.

Comment : déterminer le nom de connexion d'un utilisateur

Montre comment utiliser l'objet **My.User** pour obtenir le nom de connexion de l'utilisateur.

Exemples

Exemples de Windows Forms Visual Basic

Ces exemples illustrent des applications Windows Forms.

Voir aussi

Concepts

Comment faire en Visual Basic

Sécurité (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches de sécurité Visual Basic largement utilisées. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Sécurité et développement Visual Basic

Décrit les trois zones courantes où vous serez confronté à des problèmes de sécurité.

Sécurité dans Visual Studio

Fournit des liens et des pointeurs vers les concepts fondamentaux de la sécurité, notamment les techniques de codage sécurisé.

Notions fondamentales de la sécurité d'accès du code

Décrit la sécurité d'accès du code et ses principales utilisations.

Sécurité basée sur les rôles

Donne une vue d'ensemble de la sécurité basée sur les rôles dans .NET Framework.

Méthodes conseillées pour la stratégie de sécurité

Fournit des informations à l'attention des développeurs Windows sur les méthodes conseillées pour développer des logiciels sécurisés.

Microsoft Security Developer Center

Découvrez le travail réalisé par Microsoft pour vous aider à assurer la sécurité de vos applications et de vos systèmes (des systèmes de bureau aux systèmes réseau), avec des liens vers des ressources sur la sécurité à l'usage des professionnels de l'informatique, des développeurs et des utilisateurs privés.

Comment : modifier des connexions de base de données

Donne les conseils relatifs à des problèmes tels que ceux qui sont liés à l'enregistrement de la chaîne de connexion à une base de données.

Méthodes conseillées

Méthodes conseillées pour la stratégie de sécurité

Fournit des informations à l'attention des développeurs Windows sur les méthodes conseillées pour développer des logiciels sécurisés.

Sécurité Web

Sécurisation des Services Web XML créés à l'aide d'ASP.NET

Explique le fonctionnement de l'authentification et de l'autorisation destinées aux services Web ASP.NET.

Sécurité des sites Web ASP.NET (Visual Studio)

Les exemples fournis dans cette section montrent comment créer des applications ASP.NET incorporant différentes sortes de dispositifs de sécurité.

Limitation de l'accès aux sites Web ASP.NET

Explique comment contrôler les utilisateurs autorisés à accéder à vos fichiers projet Web ainsi que leur niveau d'accès à l'application.

Sécurité Windows

Vue d'ensemble de la sécurité dans les Windows Forms

Présente brièvement le modèle de sécurité .NET Framework et les étapes élémentaires nécessaires pour garantir la sécurité des Windows Forms dans votre application.

Impression plus sécurisée dans les Windows Forms

Explique comment accéder aux fonctionnalités d'impression dans un environnement de niveau de confiance partiel.

Accès plus sécurisé aux fichiers et aux données dans les Windows Forms

Explique comment accéder aux fichiers et aux données dans un environnement de niveau de confiance partiel.

Déploiement et sécurité ClickOnce

Examine les problèmes d'autorisation relatifs aux applications déployées au moyen de la technologie ClickOnce qui s'exécute dans le contexte d'un "bac à sable" (sandbox), avec des autorisations restreintes basées sur une zone de sécurité.

Sécurité des composants

Sécurité d'accès du code

Explique les concepts de la sécurité du code et l'approche .NET Framework de la sécurisation des composants.

Sécurité impérative

Décrit l'utilisation de vérifications de sécurité impératives pour protéger les lignes de code sensibles contre toute utilisation non autorisée.

Sécurité de base de données

Comment : modifier des connexions de base de données

Donne les conseils relatifs à des problèmes tels que ceux qui sont liés à l'enregistrement de la chaîne de connexion à une base de données.

Exemples

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Voir aussi

Concepts

Comment faire en Visual Basic

Débogage et test (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches de génération, de débogage et de test Visual Basic largement utilisées. Pour afficher d'autres catégories de tâches courantes traitées dans l'aide, consultez Comment faire en Visual Basic.

En Général

Procédure pas à pas : débogage d'un Windows Form

Illustre le débogage d'une application Windows Forms.

Procédure pas à pas : débogage d'un formulaire Web

Illustre le débogage d'un formulaire Web.

Procédure pas à pas : débogage d'un service Web XML

Illustre le débogage d'un service Web XML.

Génération d'applications

Comment : préparer et gérer des générations

Explique d'une part comment définir l'ordre de génération et les options d'enregistrement, et, d'autre part, comment générer ou régénérer une solution ou un projet.

Comment : gérer des configurations de génération en appliquant les paramètres du développeur Visual Basic

Explique comment créer et modifier les informations de configuration de génération dans Visual Basic.

Comment : configurer des projets pour des plates-formes cibles

Explique comment configurer un projet en fonction d'une plate-forme spécifique.

Débogage d'applications

Débogage de l'application Visual Basic

Fournit des pointeurs vers la documentation relative aux fonctionnalités de débogage intégrées dans Visual Studio.

Comment : démarrer l'exécution

Fournit des instructions relatives au démarrage du débogage.

Comment : effectuer un pas à pas détaillé dans le code

Explique comment exécuter le code ligne par ligne.

Comment : utiliser la fenêtre Processus

Montre comment démarrer avec la fenêtre **Processus**.

Comment : définir un point d'arrêt simple

Montre comment suspendre une application à un certain point.

Comment : utiliser la boîte de dialogue Espion express

Illustre un moyen rapide de rechercher et d'évaluer des variables et des expressions.

Comment : appliquer des modifications en mode arrêt à l'aide de la fonctionnalité Modifier & Continuer

Explique comment modifier votre code en mode arrêt, puis poursuivre sans arrêter et relancer l'exécution.

Comment : corriger des erreurs d'exécution avec l'Assistant Exception

Explique comment accéder à des conseils sur la résolution des problèmes lorsque votre code rencontre une exception.

Comment : déboguer une application de confiance partielle

Montre comment déboguer une application de confiance partielle que vous avez l'intention de déployer avec ClickOnce.

Test d'applications

Comment : compiler et exécuter un projet dans Visual Basic

Décrit vos options pour la compilation et l'exécution à partir de l'environnement de développement.

Détection et correction des erreurs du code managé

Fournit des pistes pour détecter et corriger les erreurs de code à l'aide de Microsoft Visual Studio 2005 Team Edition for Software Developers.

Surveiller des applications

Procédure pas à pas : exploration des journaux des événements, des sources d'événements et des entrées

Montre comment vos applications et vos composants peuvent enregistrer des informations à propos d'événements importants. Ces enregistrements vous permettent ensuite d'auditer les accès au système, de résoudre les problèmes éventuels et de recréer des modèles d'utilisation.

Comment : tracer le code dans une application

Décrit les principales étapes relatives à l'instrumentation et au traçage.

Comment : effectuer une compilation conditionnelle avec Trace et Debug

Explique comment inclure des options de traçage dans des applications compilées.

Voir aussi

Concepts

Comment faire en Visual Basic

Visite guidée de Visual Basic

Que vous soyez novice en Visual Basic 2005, voire même en programmation informatique, c'est ici que vous devez commencer. La Visite guidée de Visual Basic est une série de leçons séquentielles qui vous initieront aux notions de base de la programmation dans Visual Basic 2005.

Dans cette section

Mise en route de Visual Basic Express

Série des rubriques présentant Microsoft Visual Basic 2005 Express et l'environnement de développement Visual Basic.

Création de votre premier programme Visual Basic

Dans cette section, vous verrez à quel point il est facile de créer un programme pour afficher des pages Web.

Introduction au langage de programmation Visual Basic

Dans cette section, vous apprendrez les notions de base du langage Visual Basic.

Création de l'aspect visuel de votre programme : introduction aux Windows Forms

Dans cette section, vous apprendrez comment utiliser des formulaires et des contrôles pour créer une interface utilisateur.

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Dans cette section, vous apprendrez comment utiliser les outils de débogage proposés dans Visual Basic 2005.

Gestion de vos enregistrements : utilisation de données dans votre programme

Dans cette section, vous apprendrez à utiliser des bases de données.

Programmation avec des objets : utilisation de classes

Dans cette section, vous apprendrez les notions de base de la programmation orientée objet.

Objets visibles : création de votre premier contrôle utilisateur

Dans cette section, vous apprendrez comment créer votre propre contrôle réutilisable.

Dessin d'images : utilisation de graphiques

Dans cette section, vous apprendrez les possibilités graphiques de Visual Basic 2005.

Distribution de votre programme

Dans cette section, vous apprendrez comment partager vos programmes avec d'autres utilisateurs.

Étape suivante : que faire ensuite ?

Série de rubriques fournissant des liens vers des ressources supplémentaires pour la maîtrise de Visual Basic 2005.

Comment faire dans Visual Basic Express

Série de rubriques fournissant des liens vers des informations spécifiques aux tâches.

Référence

Référence Visual Basic

Décrit tous les éléments programmables dans Visual Basic

Mise en route de Visual Basic Express

Que vous soyez un utilisateur Visual Basic 6.0 expérimenté qui cherche à se mettre à niveau, un programmeur expérimenté intéressé par un premier apprentissage de Visual Basic, voire un débutant en programmation informatique, Microsoft Visual Basic 2005 Express a été conçu pour vous. Plus qu'une simple version allégée de Visual Basic, Microsoft Visual Basic 2005 Express a été expressément élaboré pour rendre la programmation plus facile, et plus amusante, que jamais.

La visite guidée de Visual Basic comprend une série des leçons qui vous mettront rapidement le pied à l'étrier à la fois avec le langage Visual Basic et les outils de Visual Basic Express. Avant de commencer, vous pouvez examiner différentes rubriques de cette section pour vous familiariser avec certaines fonctions de Visual Basic Express. Si vous débutez en programmation, vous pouvez commencer votre formation tout de suite avec Création de votre premier programme Visual Basic, puis revenir ultérieurement à ces rubriques.

Dans cette section

Introduction à Visual Basic Express

Plus qu'un simple outil d'apprentissage, Visual Basic Express fournit un environnement de développement complètement fonctionnel aux programmeurs débutants et de loisir intéressés par le développement d'applications Windows Forms, d'applications console et de bibliothèques de classes.

Vue d'ensemble de l'outil Visual Basic Express

Fournit une vue d'ensemble de Visual Basic Express, un programme permettant de créer des applications à l'aide du langage Visual Basic.

Comment : utiliser une application existante

Si vous êtes déjà familiarisé avec Visual Basic et disposez d'une application avec laquelle vous souhaitez travailler, ouvrir celle-ci est facile.

Starter Kits : pour être rapidement opérationnel

Visual Basic Express fournit des Starter Kits pour vous aider à commencer la rédaction de programmes immédiatement.

Types de projets dans Visual Basic Express

Visual Basic Express vous permet de créer plusieurs types d'applications, notamment les applications Windows Forms, les applications console et les bibliothèques de classes.

Dépannage de Visual Basic Express

Bien que Visual Basic Express facilite la programmation, il ne peut pas systématiquement se comporter comme vous le prévoyez.

Rubriques connexes

Visite guidée de Visual Basic

La Visite guidée de Visual Basic est une série de leçons séquentielles qui vous initieront aux notions de base de la programmation dans Visual Basic 2005.

Création de votre premier programme Visual Basic

Le meilleur moyen d'apprendre la programmation avec Visual Basic consiste à créer un vrai programme.

Étape suivante : que faire ensuite ?

Maintenant que vous avez fini les différentes leçons du Visite guidée de Visual Basic, il est temps de passer à autre chose.

Aide pour les utilisateurs de Visual Basic 6.0

Votre connaissance de Visual Basic 6.0 vous aidera rapidement à devenir productif avec Visual Basic 2005.

Nouveautés pour les utilisateurs de Visual Basic 6.0

Si vous êtes un utilisateur expérimenté de Visual Basic 6.0, vous trouverez dans Visual Basic 2005 de nombreuses fonctionnalités nouvelles ou considérablement améliorées, qui rendront le développement avec Visual Basic plus facile et plus puissant que dans les versions antérieures.

Introduction à Visual Basic Express

Si vous pouvez imaginer un logiciel, vous pouvez probablement le créer avec Microsoft Visual Basic 2005 Express. Du programme simple qui affiche un message à l'application complète qui accède à une base de données ou à un service Web, Visual Basic Express vous fournit les outils dont vous avez besoin.

Plus qu'un simple outil d'apprentissage, Visual Basic Express fournit un environnement de développement complètement fonctionnel aux programmeurs débutants et de loisir intéressés par le développement d'applications Windows Forms, d'applications console et de bibliothèques de classes. Les Starter kits et autres fonctionnalités d'introduction rendent de nombreuses fonctionnalités de Visual Basic accessibles aux programmeurs débutants.

Définition de Visual Basic Express

Visual Basic Express est une version à bas prix de Visual Basic. C'est à la fois un outil d'apprentissage de la programmation dans Visual Basic et un outil de développement complètement fonctionnel destiné aux programmeurs qui n'ont pas besoin de la version complète de Visual Basic. Cependant, Visual Basic Express est plus qu'un simple sous-ensemble de Visual Basic : il intègre de nombreuses fonctionnalités qui facilitent plus que jamais la programmation en Visual Basic.

Le meilleur moyen de savoir ce que vous pouvez faire avec Visual Basic Express consiste à suivre pas à pas les leçons présentées dans le Visite guidée de Visual Basic. Une fois ce programme pédagogique terminé, vous serez familiarisé avec les outils et les concepts de Visual Basic, et prêt à commencer à écrire vos propres programmes.

À qui est s'adresse Visual Basic Express ?

Visual Basic Express est un outil puissant capable de la créer des applications complètement fonctionnelles et des composants pouvant être partagés avec des tiers. Toutefois, il n'est pas destiné aux développeurs professionnels ou aux programmeurs qui travaillent dans un environnement de travail en équipe. D'autres versions de Visual Basic fournissent des fonctionnalités qui répondront aux besoins avancés des professionnels et du développement en équipe.

Si vous devez écrire des applications qui se connectent à une base de données mise en réseau, interagissent avec Microsoft Office, prennent en charge des périphériques mobiles ou des systèmes d'exploitation 64 bits, ou exigent un débogage distant, utilisez plutôt une version plus avancée de Visual Basic.

Remarque
Visual Basic Express ne prend pas en charge le développement d'applications Web ; si vous souhaitez effectuer un développement Web, vous pouvez télécharger Visual Web Developer Express.

Obtention d'aide

Les fichiers d'aide fournis avec Visual Basic Express sont un sous-ensemble de MSDN Library pour les éditions Visual Studio 2005 Express qui sont elles-mêmes un sous-ensemble de la bibliothèque MSDN Library intégrale. Si vous êtes connecté à Internet, vous pouvez accéder à toute rubrique d'aide de l'ensemble de la bibliothèque. Si vous ne disposez pas de l'accès en ligne ou si vous avez choisi de ne pas installer MSDN Library pour les éditions Visual Studio 2005 Express, il se peut que certaines rubriques d'aide ne soient pas disponibles. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Voir aussi

Concepts

Types de projets dans Visual Basic Express

Autres ressources

Mise en route de Visual Basic Express

Visite guidée de Visual Basic

Types de projets dans Visual Basic Express

Microsoft Visual Basic 2005 Express vous permet de créer plusieurs types d'applications, notamment les applications Windows Forms, les applications console et les bibliothèques de classes. Chaque application que vous créez est contenue dans un projet propre. Des modèles de projet sont fournis pour vous aider à vous lancer.

Lorsque vous créez un nouveau projet, les icônes qui figurent dans les boîtes de dialogue **Nouveau projet** et **Ajouter un projet** représentent les types de projets disponibles et leurs modèles. Les modèles de projet suivants sont disponibles lorsque vous ouvrez un nouveau projet dans Visual Basic Express.

Types de projet

Modèle de projet	Permet de créer
Modèle Application Windows	Utilisé pour créer des applications Windows qui s'exécutent localement sur l'ordinateur de l'utilisateur. Vous pouvez construire n'importe quel type de programme, de l'outil à fenêtre unique, tel que la Calculatrice de Windows, à l'application complète dotée de plusieurs fenêtres et de fonctions avancées.
Modèle Bibliothèque de classes	Utilisé pour créer des classes ou des composants réutilisables qui peuvent être partagés avec d'autres projets.
Modèle Application console	Utilisé pour créer des applications de ligne de commande, des programmes qui s'exécutent depuis une invite de commandes Windows et qui sont exempts d'interface graphique.
Modèle **Starter Kit Ma cinémathèque**	Utilisé pour créer l'application exemple prégénérée Ma cinémathèque, que vous pouvez personnaliser pour satisfaire vos propres besoins. Pour plus d'informations, consultez Starter Kits : pour être rapidement opérationnel.
Modèle **Starter Kit Écran de veille**	Utilisé pour créer l'application exemple prégénérée Écran de veille, que vous pouvez personnaliser pour satisfaire vos propres besoins. Pour plus d'informations, consultez Starter Kits : pour être rapidement opérationnel.

Remarque

Bien qu'il n'y ait aucun modèle de projet pour les bibliothèques de contrôles Windows, rien ne vous empêche de créer vos propres contrôles à l'aide du modèle **Bibliothèque de classes**. Pour plus d'informations, consultez Objets visibles : création de votre premier contrôle utilisateur.

Voir aussi

Concepts

Introduction à Visual Basic Express

Autres ressources

Visite guidée de Visual Basic

Vue d'ensemble de l'outil Visual Basic Express

Cette rubrique fournit une vue d'ensemble de Visual Basic Express, un programme permettant de créer des applications à l'aide du langage Visual Basic. Comme un programme tel que Microsoft Outlook propose divers outils pour l'utilisation de la messagerie électronique, Visual Basic Express est une trousse à outils permettant d'exécuter une large gamme de tâches de programmation.

Conseil

Si vous êtes novice en matière de programmation, vous pouvez parcourir la Visite guidée de Visual Basic, un jeu de leçons de stiné à vous apprendre les notions de base, puis revenir à cette rubrique. Pour commencer la visite, consultez Création de votre premier programme Visual Basic.

Processus de développement

Visual Basic Express facilite le processus de développement une application ; dans la plupart des cas, ce processus se compose des étapes suivantes :

- **Créez un projet.** Un projet contient tous les fichiers nécessaires à votre application, et stocke des informations sur votre application. Il arrive qu'une application contienne plusieurs projets (par exemple, un projet **Application Windows** et un ou plusieurs projets **Bibliothèque de classes**. Ce type d'application porte le nom de *solution*, qui est un synonyme de groupe de projets.
- **Concevez l'interface utilisateur.** Pour cela, vous devez faire glisser différents contrôles, tels que des boutons et des zones de texte, jusqu'à une aire de conception appelée *formulaire*. Vous pouvez ensuite définir des propriétés qui déterminent l'aspect et le comportement du formulaire ainsi que de ses contrôles.

  Remarque

  Pour les applications qui ne possèdent aucune interface utilisateur, telles que les bibliothèques de classes ou les applica tions console, cette étape n'est pas nécessaire.

- **Écrivez le code.** Ensuite, vous devrez écrire le code Visual Basic qui définit la manière dont votre application doit se comporter et interagir avec l'utilisateur. Visual Basic Express facilite l'écriture du code grâce à des fonctionnalités telles qu'Intellisense, la saisie semi-automatique et les extraits de code.
- **Testez le code.** Il est toujours conseillé de tester votre application pour vous assurer qu'elle se comporte de la manière escomptée. Ce processus est appelé *débogage*. Visual Basic Express propose des outils de débogage qui facilitent la recherche et la résolution interactives des erreurs dans votre code.
- **Distribuez l'application.** Une fois votre application terminée, vous pouvez installer le programme obtenu sur votre ordinateur ou le distribuer et le partager avec des tiers. Visual Basic Express utilise une nouvelle technologie appelée publication ClickOnce qui permet de déployer facilement une application à l'aide d'un Assistant, et de fournir automatiquement des versions mises à jour de l'application si vous apportez des modifications ultérieurement.

Exploration

À première vue, l'interface utilisateur de Visual Basic Express, également appelée *environnement de développement intégré* ou *IDE*, peut paraître peu familière, mais une fois que vous l'aurez mieux appréhendée, vous apprécierez sa convivialité. Les sections suivantes décrivent les parties de l'IDE que vous utiliserez le plus.

Au démarrage

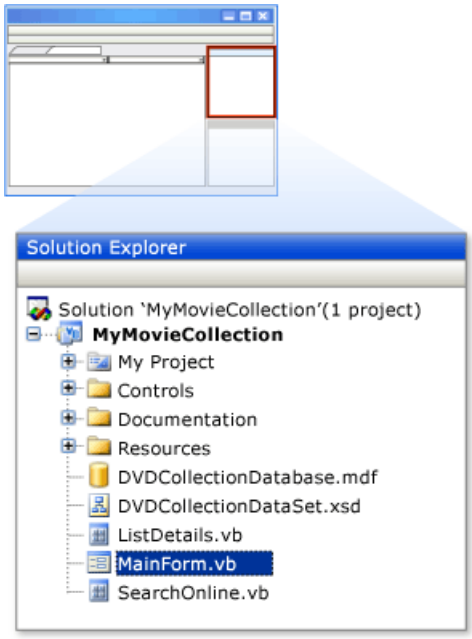
Lorsque vous ouvrez Visual Basic Express pour la première fois, vous constatez qu'une grande partie de l'IDE est remplie par la fenêtre **Page de démarrage**. La **Page de démarrage** contient une liste interactive de vos projets récents, une zone **Mise en route** fournissant des liens vers des rubriques d'aide importantes, ainsi qu'une liste de liens vers des articles en ligne et d'autres ressources. Si vous êtes connecté à Internet, cette liste est régulièrement mise à jour.

Vous pouvez modifier les informations qui s'affichent sur la **Page de démarrage** en fonction de vos préférences personnelles. Pour plus d'informations, consultez Comment : personnaliser la section Informations de la page de démarrage.

La fenêtre **Explorateur de solutions** est affichée du côté droit de l'IDE. Elle est initialement vierge, mais elle contiendra les

informations sur votre projet, ou vos groupes de projets appelés solutions. Pour plus d'informations, consultez Utilisation de l'Explorateur de solutions.

Figure 1 : Explorateur de solutions



Du côté gauche de l'IDE, vous pouvez constater la présence d'un onglet vertical marqué **Boîte à outils**. Il est aussi initialement vierge, mais au fur et à mesure que vous travaillez, il se remplit avec les éléments pouvant être utilisés pour la tâche que vous exécutez actuellement. Pour plus d'informations, consultez Utilisation de la boîte à outils.

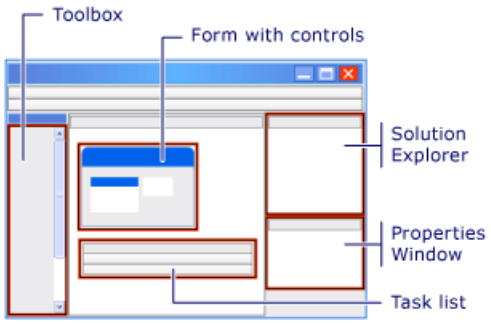
En haut de l'IDE se trouve une barre de menus et une barre d'outils. Les menus et boutons de barre d'outils disponibles changent selon la tâche en cours ; prenez le temps d'explorer et d'observer les choix disponibles. Vous pouvez également personnaliser les menus et la barre d'outils en fonction de vos préférences personnelles. Pour plus d'informations, consultez Comment : personnaliser des barres d'outils (Visual Studio).

Dans la partie inférieure de l'IDE se trouve une barre d'état indiquant **Prêt**. Au fur et à mesure que vous travaillez dans l'IDE, la barre d'état change, en affichant des messages liés à la tâche en cours (par exemple, la barre d'état peut afficher des informations sur la progression d'un projet que vous générez).

Mode Design

Lorsque vous ouvrez ou créez un projet, l'apparence de l'IDE laisse la place au *mode Design*. Il s'agit de la partie visuelle de Visual Basic, dans laquelle vous pouvez concevoir l'aspect de votre application.

Figure 2 : IDE en mode Design



En mode Design, la **Page de démarrage** est couverte par une autre fenêtre appelée **Concepteur de formulaires** qui est à la base une zone de dessin vierge représentant l'interface utilisateur de votre application. Remarquez que vous pouvez toujours accéder à la **Page de démarrage** en cliquant sur son onglet au-dessus du **Concepteur de formulaires**.

Lorsque le **Concepteur de formulaires** est visible, la **Boîte à outils** contient plusieurs contrôles (représentations de boutons, de champs de texte, de grilles, etc.) qui peuvent être placés sur le formulaire et organisés comme vous le souhaitez. Pour plus d'informations, consultez Concepteur Windows Forms.

Vous constaterez également qu'une nouvelle fenêtre, la fenêtre **Propriétés**, apparaît sous de la fenêtre de l'**Explorateur de solutions**. C'est à cet endroit que vous définirez les différentes propriétés qui définissent l'apparence et le comportement du

formulaire et de ses contrôles. Pour plus d'informations, consultez Propriétés, fenêtre.

La fenêtre Liste des tâches située en bas de l'IDE n'est pas affichée par défaut, mais constitue une zone de suivi des tâches à exécuter ou de prise de notes pendant la programmation. Pour plus d'informations, consultez Liste des tâches (Visual Studio).

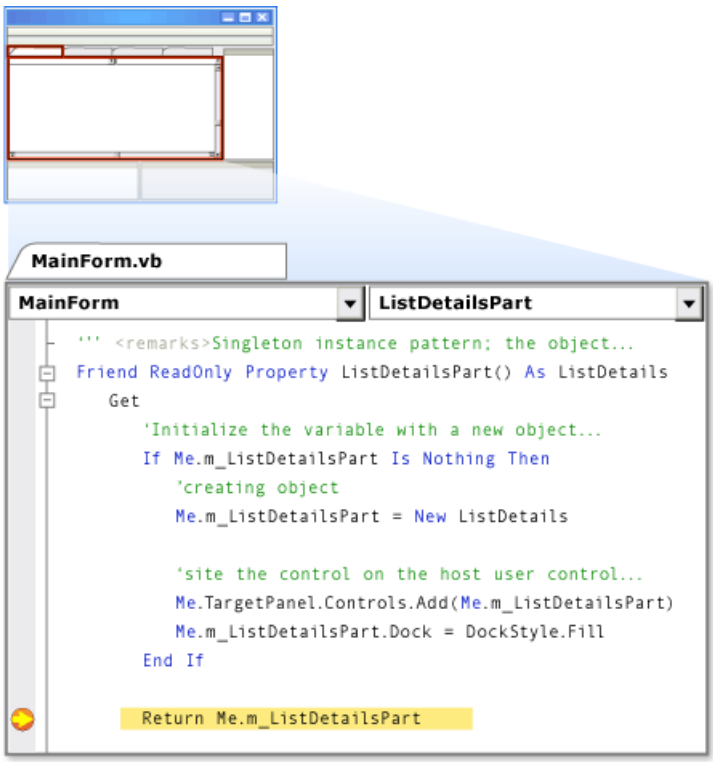
Si vous double-cliquez sur un formulaire ou un contrôle, une nouvelle fenêtre appelée éditeur de code s'ouvre. Elle vous permet d'écrire le code réel de votre application. Plus qu'un simple éditeur de texte, l'éditeur de code utilise une technologie appelée Intellisense pour vous aider à écrire du code en fournissant des informations pertinentes au fur et à mesure que vous tapez. Pour plus d'informations, consultez Options IntelliSense spécifiques à Visual Basic.

Remarque
Pour certains types de projets, tels que les projets **Bibliothèque de classes** qui n'ont aucune interface utilisateur, l'éditeur d e code s'affiche plutôt que le **Concepteur de formulaires**.

Mode exécution

Lorsque vous exécutez ou déboguez votre application, l'IDE passe en *mode exécution*. L'application proprement dite est lancée et une fenêtre supplémentaire liée au débogage s'affiche. En mode exécution, vous ne pouvez pas apporter de modification dans le **Concepteur de formulaires**, la fenêtre **Propriétés** ou l'**Explorateur de solutions**, mais vous pouvez apporter des modifications à votre code dans l'éditeur de code.

Figure 3 : Formulaire Visual Basic en mode arrêt



En mode exécution, une nouvelle fenêtre appelée fenêtre **Exécution** s'affiche en bas de l'IDE. Si vous mettez l'application en mode arrêt, vous pouvez interroger des valeurs et tester le code dans la fenêtre **Exécution**. Pour plus d'informations, consultez Fenêtre Exécution.

Des fenêtres supplémentaires pour permettant de consulter les valeurs des variables, d'afficher les résultats, et d'exécuter d'autres tâches de débogage peuvent s'afficher au moment de l'exécution si vous les sélectionnez dans le menu **Déboguer**.

Autres fenêtres importantes

Vous pouvez rencontrer de nombreuses autres fenêtres dans l'IDE, chacune étant axée sur une tâche de programmation spécifique. Vous trouverez ci-dessous la liste des fenêtres les plus courantes.

- La fenêtre **Liste d'erreurs** apparaît en bas de l'IDE si vous entrez du code incorrect ou si d'autres erreurs se produisent au moment de la conception. Pour plus d'informations, consultez Liste d'erreurs, fenêtre.
- La fenêtre **Explorateur d'objets** permet d'examiner les propriétés, méthodes et événements de tout objet pouvant être utilisé dans votre application. Pour plus d'informations, consultez Explorateur d'objets.

- Le **Concepteur de projets** permet de configurer les propriétés de votre application, notamment les ressources, le comportement de débogage, les paramètres de déploiement, etc. Pour plus d'informations, consultez Introduction au Concepteur de projets.
- L'**Explorateur de bases de données** permet d'afficher et d'utiliser les bases de données existantes ou d'en créer et d'en concevoir de nouvelles. Pour plus d'informations, consultez Explorateur de serveurs/Explorateur de bases de données.

Personnalisation

Visual Basic Express permet de personnaliser l'IDE en réorganisant la disposition des fenêtres, en choisissant celles qui doivent être affichées, en ajoutant ou en supprimant des commandes de menu et des boutons de barre d'outils, etc. Pour plus d'informations, consultez Personnalisation de l'environnement de développement.

Obtention d'aide

Quand vous utilisez Visual Basic Express, l'aide est toujours à portée de main. Si vous vous trouvez dans l'éditeur de code ou dans toute autre fenêtre, appuyez sur la touche F1 pour afficher la rubrique d'aide correspondant le mieux à l'opération que vous effectuez à ce moment. Par exemple, si vous êtes dans l'éditeur de code et si le curseur se trouve sur le mot clé **Inherits**, le navigateur d'aide démarre et une rubrique décrivant l'utilisation de l'instruction **Inherits** s'affiche.

Remarque
Les fichiers d'aide fournis avec Visual Basic Express sont un sous-ensemble de MSDN Library pour les éditions Visual Studio 2005 Express qui sont elles-mêmes un sous-ensemble de la bibliothèque MSDN Library intégrale. Si vous êtes connecté à Internet, vous pouvez accéder à toute rubrique d'aide de l'ensemble de la bibliothèque. Si vous ne disposez pas de l'accès en ligne ou si vous avez choisi de ne pas installer MSDN Library pour les éditions Visual Studio 2005 Express, il se peut que certaines rubriques d'aide ne soient pas disponibles.

L'aide peut également être lancée à partir du menu **?** (Aide), et vous pouvez rechercher les rubriques souhaitées à l'aide des fenêtres **Sommaire** ou **Index**, ou de l'onglet **Rechercher**. Pour plus d'informations, consultez Aide sur l'aide (aide de Microsoft Document Explorer).

Voir aussi

Autres ressources

Création de votre premier programme Visual Basic
Visite guidée de Visual Basic
Aide pour les utilisateurs de Visual Basic 6.0
Guide de programmation Visual Basic
Création d'applications Windows

Comment : utiliser une application existante

Si vous êtes déjà familiarisé avec Visual Basic et disposez d'une application avec laquelle vous souhaitez travailler, ouvrir celle-ci est facile.

Pour ouvrir une application existante

1. Dans le menu **Fichier**, cliquez sur **Ouvrir un projet**.
2. Dans la boîte de dialogue **Ouvrir un projet**, sélectionnez le projet que vous souhaitez ouvrir, puis cliquez sur **Ouvrir**.

Voir aussi

Concepts

Utilisation de l'Explorateur de solutions

Autres ressources

Gestion des propriétés de projet avec le Concepteur de projets

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Starter Kits : pour être rapidement opérationnel

Considérez un Starter Kit comme un modèle ou un programme modèle qui permet à la fois de connaître de nouveaux outils et de nouvelles techniques de programmation, et de construire facilement un programme opérationnel. Les Starter Kits incluent des exemples de code, de la documentation et d'autres ressources utiles.

Vous pouvez utiliser un Starter Kit tel quel ou le personnaliser. Vous pouvez le partager facilement avec des amis et des collègues, ou avec d'autres membres de la communauté Visual Basic. Tous les Starter Kits Microsoft Visual Basic 2005 Express comprennent des didacticiels qui vous aident à les personnaliser. De plus, l'ensemble du code comprend des commentaires qui vous aideront à comprendre son fonctionnement.

Starter Kits Visual Basic

Visual Basic Express inclut deux Starter Kits. Vous trouverez des Starter Kits Visual Basic dans la boîte de dialogue **Nouveau projet**, accessible depuis le menu **Fichier**. Sélectionnez celui que vous voulez et mettez-vous au travail !

Starter Kit : Ma cinémathèque

Le Starter Kit Ma cinémathèque est un exemple d'application conçu pour vous aider à organiser votre collection de films et à consulter des informations en ligne. L'application est fortement personnalisable ; vous pouvez la personnaliser et la modifier pour organiser une autre collection.

Une fois généré, le Starter Kit Ma cinémathèque permet d'effectuer les tâches suivantes :

- Intégrer des informations descriptives sur votre collection de films.
- Stocker et enregistrer les informations dans une base de données.
- Rechercher en ligne des informations sur vos films.

Starter Kit : Écran de veille

Le Starter Kit Écran de veille vous aide à créer un écran de veille Windows qui affiche sur votre ordinateur des flux RSS Internet en direct. Le projet est fourni prêt à compiler et à exécuter, mais il est facile à personnaliser.

Il facilite notamment les personnalisations suivantes :

- Sélection de votre propre image par défaut intégrée
- Définition de plusieurs images d'arrière-plan.
- Modification de la liste des flux RSS.
- Construction de votre propre écran de veille.
- Utilisation des services Web XML pour afficher des informations ; par exemple, la météo.

Mise en route : lancement de votre Starter Kit

La première étape de la mise en œuvre de votre Starter Kit consiste à le charger. Vous pouvez ensuite exécuter l'application et examiner quelques-unes des tâches simples qu'elle vous permet d'effectuer.

Pour charger et exécuter l'application

1. Dans le menu **Fichier**, cliquez sur **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, cliquez sur le Starter Kit de votre choix dans la liste **Modèles Visual Studio installés**, puis sur **OK**.

   Le projet du Starter Kit est chargé.

Remarque
Vous pouvez modifier le nom de votre version de l'application dans la zone **Nom**.

3. Appuyez sur F5 pour exécuter l'application.
4. Suivez les étapes du didacticiel du Starter Kit pour comprendre et personnaliser votre application.

Voir aussi

Référence

Nouveau projet, boîte de dialogue

Concepts

Comment : créer des Starter Kits

Autres ressources

Starter Kits

Dépannage de Visual Basic Express

Bien que Visual Basic Express facilite la programmation, il se peut qu'il ne se comporte pas systématiquement comme vous le prévoyez. À mesure que vous parcourez la Visite guidée de Visual Basic ou que vous écrivez vos propres programmes, vous trouverez sans doute ici l'aide nécessaire pour résoudre les problèmes que vous êtes susceptible de rencontrer.

Une leçon de la visite guidée ne fonctionne pas

Au fur et à mesure que vous parcourez les leçons de la visite guidée, il se peut que vous constatiez que votre programme ne s'exécute pas ou qu'il ne se comporte pas de la manière escomptée. Dans la plupart des cas, ces erreurs se produisent parce que vous avez ignoré une étape ou fait une faute de frappe dans le code.

Conseil
Lors de la saisie du code d'une leçon de la visite guidée, utilisez le bouton **Copier le code** de la rubrique d'aide et collez le co de dans l'éditeur de code. C'est plus simple que de le taper et les risques d'erreurs sont moindres.

Il est également possible que des étapes ou du code fournis dans la leçon soient erronés en raison de modifications de dernière minute apportées au produit. Les versions en ligne de ces rubriques sont mises à jour en permanence ; il se peut donc qu'une version corrigée de la rubrique soit disponible en ligne.

Pour corriger les erreurs dans une leçon

1. Revenez à la dernière étape à laquelle le programme fonctionnait et suivez minutieusement chaque étape restante.
2. Examinez tout code que vous avez saisi et vérifiez qu'il correspond exactement au code de la leçon.
3. Appuyez sur F5 pour exécuter le programme.

   Si le programme ne s'exécute toujours pas ou ne se comporte toujours pas de la manière escomptée, vérifiez si une version mise à jour de la rubrique est disponible. Pour plus d'informations, consultez la section « Impossible d'accéder à une rubrique d'aide » ci-dessous.

Je ne parviens pas à trouver une commande ou une option.

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Impossible d'accéder à une rubrique d'aide

Les fichiers d'aide fournis avec Visual Basic Express sont un sous-ensemble de MSDN Library pour les éditions Visual Studio 2005 Express qui sont elles-mêmes un sous-ensemble de la bibliothèque MSDN Online Library intégrale. Lors de l'installation, vous avez pu choisir d'installer MSDN Library pour les éditions Visual Studio 2005 Express (si vous avez choisi de ne pas l'installer, faites-le maintenant).

Un lien ne fonctionne pas

Les liens vers des rubriques qui ne sont pas fournies dans MSDN Library pour les éditions Visual Studio 2005 Express affichent la page **Information introuvable**, mais vous pouvez toujours accéder aux rubriques auxquels les liens font référence en parcourant MSDN Online Library.

Pour accéder à une rubrique lorsqu'un lien ne fonctionne pas

1. Dans le menu **?** (Aide), cliquez sur **Rechercher**.

   La page **Rechercher** s'ouvre.
2. Dans la zone de recherche, entrez le texte qui était affiché dans le lien, puis cliquez sur **Rechercher**.

   Une recherche est lancée et les résultats s'affichent.
3. Sélectionnez l'onglet **MSDN Online** du côté droit de la page **Rechercher**.

Remarque

Si **MSDN Online** n'est pas affiché, il se peut que votre système d'aide ne soit pas configuré pour l'accès en ligne. Pour plus d'informations, consultez la section « Comment : activer l'accès en ligne » ci-dessous.

4. Cliquez sur le titre de la rubrique qui correspond le mieux au texte du lien d'origine.

   La rubrique est téléchargée et affichée.

L'appui de la touche F1 ne fonctionne pas

Si vous appuyez sur F1 à n'importe quel endroit de l'IDE, une aide contextuelle doit s'afficher. Même si toutes les rubriques ne sont pas incluses dans MSDN Library pour les éditions Visual Studio 2005 Express, si vous êtes connectés à Internet, vous pouvez accéder à toute rubrique d'aide de l'ensemble de la bibliothèque à l'aide de la touche F1. Si vous ne disposez pas de l'accès en ligne ou si vous avez choisi de ne pas activer l'aide en ligne, il se peut que certaines rubriques d'aide ne soient pas disponibles avec la touche F1.

Vous pouvez modifier à tout moment les options de l'aide comme décrit ci-dessous.

Comment : activer l'accès en ligne

La première fois que vous accédez à l'aide à partir de Visual Basic Express, vous êtes invité à choisir la configuration de l'aide en ligne. Vous pouvez choisir d'essayer d'abord de rechercher la version en ligne d'une rubrique, d'essayer d'abord de détecter l'aide installée localement et de la rechercher en ligne si elle est introuvable, ou encore d'utiliser l'aide locale uniquement.

Conseil

Si vous avez accès à Internet, il est conseillé de configurer l'aide afin de rechercher d'abord l'aide en ligne. Si une version plus récente d'une rubrique est disponible, vous disposerez toujours des informations les plus à jour.

Vous pouvez modifier à tout moment les options de l'aide comme décrit ci-dessous.

Remarque

Pour accéder à l'aide en ligne, vous devez disposer d'une connexion active à Internet et tout serveur proxy ou pare-feu doit être configuré pour autoriser l'accès à MSDN Online Library.

Pour activer l'accès en ligne

1. Si ce n'est pas encore fait, ouvrez Visual Basic à partir du menu **Démarrer** de Windows.
2. Dans le menu **Outils**, cliquez sur **Options**.
3. Dans la boîte de dialogue **Options**, activez la case à cocher **Afficher tous les paramètres**.
4. Développez le nœud **Aide** sous **Environnement**, , puis cliquez sur **En ligne**.
5. Dans la zone **Lors du chargement du contenu d'aide**, cliquez sur **Essayer d'utiliser le contenu en ligne avant le contenu local** ou sur **Essayer d'utiliser le contenu local avant le contenu en ligne**.

   Vous devriez à présent être en mesure d'accéder à l'aide en ligne.

Ma base de données ne reflète pas les modifications

Un fichier de base de données local peut être inclus en tant que fichier dans un projet. La première fois que vous connectez votre application à un fichier de base de données local, vous pouvez choisir entre créer une copie de la base de données dans votre projet ou vous connecter au fichier de base de données existant à son emplacement actuel.

Si vous choisissez de créer une copie de la base de données, selon les paramètres du fichier, il est possible qu'une nouvelle copie de la base de données soit créée à chaque exécution du projet, remplaçant la version qui contient vos modifications. Pour plus d'informations, consultez Comment : gérer des fichiers de données locaux dans votre projet.

Voir aussi

Tâches

Résolution des problèmes d'accès aux données dans Visual Studio

Dépannage des types de données

Concepts

Dépannage du déploiement ClickOnce
Ressources pour résoudre les erreurs d'environnement de développement

Création de votre premier programme Visual Basic

Le meilleur moyen d'apprendre la programmation avec Visual Basic consiste à créer un vrai programme. Les exercices suivants vous guideront pas à pas au fil du processus de création d'un programme destiné à afficher des pages Web.

Si vous ne comprenez pas tout immédiatement, ne vous inquiétez pas ; les concepts présentés ici seront examinés en détail dans d'autres sections du Visite guidée de Visual Basic.

Dans cette section

Étape 1 : créer un projet dans Visual Basic

Étape 2 : créer une interface utilisateur

Étape 3 : personnaliser l'apparence et le comportement

Étape 4 : ajouter du code Visual Basic

Étape 5 : exécuter et tester votre programme

Rubriques connexes

Présentation détaillée : explication des propriétés, des méthodes et des événements

Types de projets dans Visual Basic Express

Introduction au langage de programmation Visual Basic

Création de l'aspect visuel de votre programme : introduction aux Windows Forms

Centre de ressources Visual Basic Express (http://msdn.microsoft.com/vbasic/express)

Étape 1 : créer un projet dans Visual Basic

La première étape pour créer un programme Visual Basic consiste à ouvrir Visual Studio et à créer un *projet*. Vous procédez ainsi pour créer n'importe quel programme Visual Basic.

Pour créer un projet pour votre programme

1. Dans le menu **Démarrer** de Windows, sélectionnez Microsoft Visual Basic 2005 Express.

   L'écran « Bienvenue dans Visual Basic Express » s'affiche. Il s'agit de l'interface pour Visual Basic 2005 Express, également connue sous le nom d'*environnement de développement intégré*, ou *IDE* (Integrated Development Environment).

2. Dans le menu **Fichier**, cliquez sur **Nouveau projet**.

   La boîte de dialogue **Nouveau projet** s'affiche.

3. Sélectionnez **Application Windows**, puis cliquez sur **OK**.

   Un nouveau *formulaire* s'affiche dans l'IDE, et les fichiers nécessaires à votre projet sont ajoutés dans la fenêtre de l'**Explorateur de solutions**. S'il s'agit du premier projet **Application Windows** que vous créez, il se nommera « WindowsApplication1 ».

Présentation détaillée

Vous venez de créer un projet pour votre programme de navigation Web. Dans Visual Basic, un projet est un emplacement où sont enregistrés et organisés les composants de votre programme.

Lorsque vous créez un projet, celui-ci n'existe d'abord qu'en mémoire. Si vous fermez l'IDE Visual Basic, vous êtes invité à enregistrer ou à ignorer le projet. Lorsque vous l'enregistrez, vous pouvez lui donner un nom plus explicite.

Une fois la boîte de dialogue **Nouveau projet** ouverte, vous avez le choix entre différents types de projet. Votre programme de navigation Web est une *application Windows* normale ; il s'agit d'un programme qui peut s'exécuter à partir de votre menu **Démarrer**.

Lorsque vous avez créé le projet, un formulaire (également connu sous le nom de *Concepteur de formulaires*) s'est affiché dans l'IDE. Ce formulaire consiste en une fenêtre qui s'affichera lorsque le programme sera exécuté. Nombre de programmes affichent plusieurs fenêtres ; un projet peut donc contenir plusieurs formulaires.

Étapes suivantes

Dans la leçon suivante, vous apprendrez comment ajouter des contrôles au formulaire pour générer votre application de navigation Web.

Leçon suivante : Étape 2 : créer une interface utilisateur

Voir aussi

Concepts

Types de projets dans Visual Basic Express

Autres ressources

Création de votre premier programme Visual Basic

Étape 2 : créer une interface utilisateur

Il est à présent temps de commencer à construire votre navigateur Web. Vous utiliserez Microsoft Visual Basic 2005 Express pour construire l'*interface utilisateur* (la partie visible avec laquelle l'utilisateur interagit) en ajoutant des *contrôles* de la **Boîte à outils** au formulaire.

La **Boîte à outils** se trouve du côté gauche de Visual Studio et se compose de plusieurs onglets tels que **Données**, **Composants** et **Tous les Windows Forms**. Chaque onglet comprend un jeu d'éléments qui représentent des contrôles ou des composants que vous pouvez ajouter à votre application. Par exemple, l'onglet **Tous les Windows Forms** possède des entrées nommées **TextBox**, **Button** et **CheckBox** représentant des contrôles que vous pouvez ajouter à votre application en les faisant glisser jusqu'au formulaire.

Pour ajouter des contrôles à votre application

1. Cliquez sur le panneau **Boîte à outils**.

   La **Boîte à outils** s'ouvre.

Conseil
La **Boîte à outils** est plus facile à utiliser si vous laissez la fenêtre ouverte. Pour ce faire, cliquez sur l'icône **Masquer a utomatiquement** qui ressemble à une punaise.

2. Cliquez l'onglet **Tous les Windows Forms** de la **Boîte à outils**, sélectionnez le contrôle **Panel**, puis faites glisser un panneau jusqu'au coin supérieur gauche du formulaire.

Conseil
Si vous ne parvenez pas à trouver le contrôle approprié, cliquez avec le bouton droit sur la **Boîte à outils**, puis sélectio nnez **Trier les éléments par ordre alphabétique**.

3. Dans le même onglet, faites glisser un contrôle **Button** et placez-le au-dessus du contrôle **Panel**.

Conseil
Vous pouvez également repositionner les contrôles par glisser-déplacer. Vous pouvez également redimensionner les c ontrôles en cliquant dessus et en faisant glisser leur bord ou leur coin.

4. Dans le même onglet, faites glisser un contrôle **TextBox** et placez-le au-dessus du contrôle **Panel**.
5. Enfin, dans l'onglet **Tous les Windows Forms**, sélectionnez un contrôle **WebBrowser** et placez-le sous le contrôle **Panel**.

Conseil
Si la fenêtre de la **Boîte à outils** est restée ouverte, vous pouvez à présent la fermer pour augmenter votre espace de t ravail. Pour ce faire, cliquez à nouveau sur l'icône **Masquer automatiquement**.

Présentation détaillée

Vous venez d'ajouter quatre contrôles au formulaire. Les contrôles contiennent du code qui définit l'aspect des contrôles et les tâches qu'ils peuvent effectuer.

Par exemple, le contrôle **Button** : pratiquement tous les programmes disposent d'un bouton OK ou d'un bouton Quitter. Vous pouvez écrire votre propre code pour dessiner un bouton à l'écran, pour modifier son apparence suite à un clic et pour qu'il exécute certaines tâches lorsqu'il est activé ; toutefois, ce développement s'avère rapidement pénible. Le contrôle **Button** contient déjà le code nécessaire à ces tâches, et vous épargne ainsi beaucoup de travail inutile.

Comme vous pouvez le constater, la **Boîte à outils** contient de nombreux contrôles, et chacun a une fonction unique. Les contrôles **Panel** peuvent être utilisés pour contenir d'autres contrôles, tels que ceux que vous venez d'ajouter. Les contrôles **Button** permettent en général d'effectuer des tâches lorsque l'utilisateur clique dessus (par exemple, fermer le programme).

Les contrôles **TextBox** sont utilisés pour saisir un texte à l'écran au moyen du clavier. Un contrôle **WebBrowser** intègre des fonctions de navigation Web semblables à celles d'Internet Explorer ; vous n'avez aucun intérêt à écrire tout le code nécessaire pour obtenir le même résultat !

Dans les leçons suivantes, vous apprendrez comment personnaliser l'aspect de ces contrôles et de nombreux autres, et comment écrire un code qui définit leur comportement. En plus des contrôles de la **Boîte à outils**, vous pouvez créer vos propres contrôles, connus sous le nom de *contrôles utilisateur* ; ils seront examinés dans une prochaine leçon.

Étapes suivantes

Vous venez d'ajouter tous les contrôles nécessaires à votre application. Son aspect autant que sa nature sont encore rudimentaires. Dans la leçon suivante, vous utiliserez la fenêtre **Propriétés** pour définir les propriétés qui contrôlent l'aspect et l'ergonomie de votre application.

Leçon suivante : Étape 3 : personnaliser l'apparence et le comportement

Voir aussi

Tâches

Étape 1 : créer un projet dans Visual Basic

Autres ressources

Création de votre premier programme Visual Basic
Contrôles à utiliser dans les Windows Forms

Étape 3 : personnaliser l'apparence et le comportement

Dans la leçon précédente, vous avez créé une interface utilisateur en ajoutant des contrôles à votre application. Toutefois, à ce stade, ni son fonctionnement ni son aspect ne semblent être ceux d'une application finie. Dans cette leçon, vous définirez des *propriétés* pour gérer l'aspect de vos contrôles, à l'aide de la fenêtre **Propriétés**.

Pour définir les propriétés de vos contrôles

1. Dans le **Concepteur de formulaires**, sélectionnez le contrôle Panel.

   La fenêtre **Propriétés** placée dans le coin inférieur droit de l'IDE affiche toutes les propriétés destinées au contrôle **Panel** nommé **Panel1**.

2. Dans la fenêtre **Propriétés**, sélectionnez la propriété Dock, puis cliquez sur la flèche de droite. Une petite fenêtre de sélection de propriétés contenant plusieurs zones s'affiche.

Conseil
La propriété **Dock** se trouve sous la catégorie **Disposition**. Vous pouvez trier alphabétiquement les propriétés en cliquant sur le bouton **AZ** dans la fenêtre **Propriétés**.

3. Cliquez sur la zone supérieure dans la fenêtre de sélection de propriétés pour attribuer à la propriété **Dock** la valeur **Top**. Le contrôle **Panel** se développe pour occuper le haut du formulaire.

4. Dans le **Concepteur de formulaires**, sélectionnez le contrôle WebBrowser. Dans la fenêtre **Propriétés**, affectez à sa propriété **Dock** la valeur **Fill** en sélectionnant la propriété **Dock**, en cliquant sur la flèche de droite et en sélectionnant la zone centrale dans la fenêtre de sélection de propriétés.

5. Dans le **Concepteur de formulaires**, sélectionnez le contrôle Button.

6. Dans la fenêtre **Propriétés**, sélectionnez la propriété Text du contrôle **Button**. Dans la colonne droite, supprimez `Button1` et remplacez-le par `Go!`.

7. Redimensionnez ou déplacez éventuellement des contrôles et redimensionnez le formulaire selon vos goûts.

Remarque
Les contrôles TextBox et **Button** doivent rester au-dessus du **Panel**. Dans la négative, vous ne pourrez pas les voir lors que vous exécuterez l'application.

Présentation détaillée

Dans cette leçon, vous avez défini plusieurs *propriétés* qui ont modifié l'aspect des contrôles destinés à votre application. Dans Visual Basic, une propriété représente un attribut d'un objet (dans ce cas, un contrôle). Par exemple, le texte qu'un contrôle **Button** affiche constitue un de ses attributs. Dans ce cas-ci, vous définissez la propriété **Text** pour qu'elle affiche « Go! ». Pour en savoir plus sur les propriétés, consultez Présentation détaillée : explication des propriétés, des méthodes et des événements.

Parallèlement au texte, les propriétés peuvent accepter de nombreux types de valeurs. Par exemple, la propriété **Dock** utilisait une fenêtre de sélection de propriétés pour afficher les options disponibles. Des nombres, une option que vous sélectionnez dans une liste, ou une option vrai/faux comptent comme autant d'autres valeurs de propriété.

Si vous avez redimensionné ou repositionné un contrôle, vous avez ici encore défini des propriétés. Les propriétés Size et Location déterminent la taille et l'emplacement d'un contrôle sur le formulaire. Pour examiner ce thème en pratique, sélectionnez la propriété **Size** dans la fenêtre **Propriétés**, puis utilisez la souris pour redimensionner le contrôle. Lorsque vous relâchez le bouton de la souris, les nouvelles valeurs **Size** s'affichent dans la fenêtre **Propriétés**.

Si la plupart des propriétés peuvent être définies dans la fenêtre **Propriétés**, elles peuvent également l'être par l'écriture d'un code. Vous en saurez plus sur l'écriture d'un code pour définir des propriétés dans une leçon suivante.

Étapes suivantes

Votre interface utilisateur est désormais terminée ! Il ne vous reste plus qu'à ajouter les instructions (appelées également *code*) pour spécifier le fonctionnement de votre programme. Dans la leçon suivante, vous ajouterez une ligne de code à votre programme.

Leçon suivante : Étape 4 : ajouter du code Visual Basic

Voir aussi

Tâches

Étape 1 : créer un projet dans Visual Basic

Étape 2 : créer une interface utilisateur

Concepts

Présentation détaillée : explication des propriétés, des méthodes et des événements

Autres ressources

Création de votre premier programme Visual Basic

Présentation détaillée : explication des propriétés, des méthodes et des événements

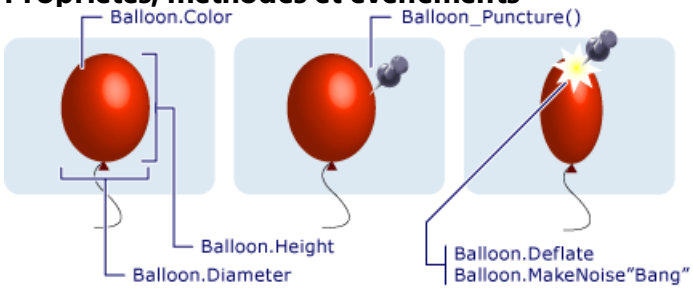
Tous les objets du langage Visual Basic, y compris les formulaires et les contrôles, disposent de leurs propres propriétés, méthodes et événements. Les propriétés peuvent être considérées comme les attributs d'un objet, les méthodes comme ses actions, et les événements comme ses réactions.

Un objet ordinaire tel qu'un ballon gonflé à l'hélium dispose également de propriétés, de méthodes et d'événements. Les propriétés du ballon comprennent des attributs visibles, tels que la hauteur, le diamètre et la couleur. D'autres propriétés décrivent son état (gonflé ou dégonflé), ou différents attributs invisibles, tels que son âge. Ces propriétés sont communes à tous les ballons, même si leurs valeurs peuvent diverger d'un ballon à l'autre.

Un ballon dispose également de méthodes ou d'actions identifiées qu'il peut exécuter. Il dispose d'une méthode de gonflage (le remplissage à l'hélium), d'une méthode de dégonflage (le vidage) et d'une méthode de décollage (le lâché). Ici encore, ces méthodes sont communes à tous les ballons.

Les ballons affichent également des réactions à certains événements externes. Par exemple, un ballon réagit à l'événement crevaison en se dégonflant, ou à l'événement lâché en s'élevant dans les airs.

Propriétés, méthodes et événements



Une ballon possède des propriétés (Color, Height et Diameter), répond à des événements (Puncture) et peut exécuter des méthodes (Deflate, MakeNoise).

Propriétés

Si vous pouviez programmer un ballon, son code Visual Basic ressemblerait au « code » suivant, qui définit les propriétés d'un ballon.

```
Balloon.Color = Red

Balloon.Diameter = 10

Balloon.Inflated = True
```

Remarquez l'ordre du code : l'objet (Balloon), suivi par la propriété (Color), suivie par l'assignation de la valeur (= Red). Vous pourriez modifier la couleur du ballon en remplaçant sa valeur.

Méthodes

Les méthodes d'un ballon sont *appelées* de la manière suivante :

```
Balloon.Inflate

Balloon.Deflate

Balloon.Rise(5)
```

L'ordre est similaire à celui d'une propriété : l'objet (un nom), suivi de la méthode (un verbe). Dans la troisième méthode, un élément supplémentaire, appelé *argument*, spécifie la hauteur d'élévation du ballon. Certaines méthodes auront un ou plusieurs arguments pour décrire plus avant l'action à effectuer.

Événements

Le ballon peut réagir à un événement de la manière suivante :

```
Sub Balloon_Puncture()
    Balloon.MakeNoise("Bang")
    Balloon.Deflate
    Balloon.Inflated = False
```

```
    End Sub
```

Dans le cas présent, le code décrit le comportement du ballon lorsqu'un événement de crevaison Puncture se produit : appelez la méthode sonore MakeNoise avec un argument "Bang", (le type de bruit à produire), puis appelez la méthode de dégonflage Deflate. Etant donné que le ballon est dégonflé, la valeur de la propriété de gonflage Inflated est **False**.

Si un ballon ne se programme pas, vous pouvez cependant programmer un formulaire ou un contrôle Visual Basic. En tant que programmeur, vous êtes responsable de l'opération. Vous devez déterminer les propriétés qui doivent être modifiées, les méthodes qui doivent être appelées ou les événements auxquels il convient de répondre pour obtenir l'apparence et le comportement souhaités.

Étapes suivantes

Dans la leçon suivante, vous ajouterez une ligne de code à votre programme.

Leçon suivante : Étape 4 : ajouter du code Visual Basic.

Voir aussi

Autres ressources

Guide de programmation Visual Basic

Étape 4 : ajouter du code Visual Basic

Dans la leçon précédente, vous avez utilisé la fenêtre **Propriétés** pour configurer les propriétés des contrôles présents sur votre formulaire. Dans cette leçon, vous ajouterez le *code* qui contrôlera les fonctions de votre programme.

Pour ajouter un code et des fonctionnalités à votre programme

1. Dans le **Concepteur de formulaires**, double-cliquez sur le contrôle Button.

   Une nouvelle fenêtre appelée l'éditeur de code s'affiche. Elle permet d'ajouter tout le code destiné à votre programme.

2. Dans l'éditeur de code, saisissez les éléments suivants.

   VB

   ```
   WebBrowser1.Navigate(Textbox1.Text)
   ```

   Ce code *s'exécute* lorsque les utilisateurs cliquent sur le bouton.

   Conseil

   Lorsque l'Éditeur de code s'ouvre, le curseur est placé automatiquement à l'intérieur de la procédure Button - vous pouvez alors commencer à taper.

Présentation détaillée

Vous avez pu remarquer que lorsque l'éditeur de code est ouvert, il contient déjà un code qui ressemble à ce qui suit :

```
Private Sub Button1_Click(ByVal sender As System.Object...

|

End Sub
```

Ce code est un *gestionnaire d'événements*, également appelé procédure *Sub*. Tout code présent dans cette procédure (entre **Sub** et **End Sub**) s'exécute à chaque fois que le bouton est activé. Vous avez pu également remarquer que le curseur se trouvait à l'intérieur de la procédure événementielle, et qu'il ne vous restait donc plus qu'à saisir votre code.

Le code que vous avez saisi (`WebBrowser1.Navigate(TextBox1.Text)`) demande au programme d'utiliser la *méthode* **Navigate** du contrôle **WebBrowser** (nommée WebBrowser1) avec un *argument* de **TextBox1.Text** (la valeur contenue dans la propriété **Text** du contrôle **TextBox**). Pour en savoir plus sur les propriétés, les méthodes, et les événements, consultez Présentation détaillée : explication des propriétés, des méthodes et des événements

Si vous ne comprenez pas ce code, ne vous inquiétez pas. Vous en saurez plus sur la rédaction du code dans les leçons suivantes.

Étapes suivantes

Votre application est à présent terminée ! Dans la leçon suivante, vous exécuterez votre première application Visual Basic.

Leçon suivante : Étape 5 : exécuter et tester votre programme

Voir aussi

Tâches

Étape 1 : créer un projet dans Visual Basic
Étape 2 : créer une interface utilisateur
Étape 3 : personnaliser l'apparence et le comportement

Concepts

Présentation détaillée : explication des propriétés, des méthodes et des événements

Autres ressources

Création de votre premier programme Visual Basic

Étape 5 : exécuter et tester votre programme

Maintenant que votre programme est terminé, il est temps de l'exécuter et de le tester. Pour les programmes complexes, la phase de test peut constituer un processus long et difficile, qui sera examiné en détail dans une leçon ultérieure. Heureusement, pour ce programme, il vous suffit de l'exécuter !

Pour exécuter votre programme

1. Connectez votre ordinateur à Internet.
2. Dans le menu **Déboguer** de l'IDE Visual Basic, cliquez sur **Démarrer le débogage**.

   Cette commande exécute votre programme.

Conseil
Pour exécuter votre programme, vous pouvez utiliser la touche F5 en tant que raccourci.

3. Dans la zone de texte, tapez `http://www.microsoft.com/france` et cliquez sur le bouton **Go!**.

   Le contrôle WebBrowser de votre programme navigue jusqu'à la page d'accueil du site Web de Microsoft. Depuis cette page, vous pouvez poursuivre la navigation par le biais de tous les liens connexes. Pour visiter une autre page Web, tapez l'adresse dans la zone de texte, puis cliquez sur le bouton **Go!**.

4. Pour fermer le programme, cliquez sur **Arrêter le débogage** dans le menu **Déboguer**.

Conseil
Vous pouvez également mettre fin au programme, en cliquant sur le bouton **Fermer** placé dans le coin supérieur droit du formulaire.

Présentation détaillée

Dans cette leçon, vous avez exécuté votre programme pour vous assurer qu'il fonctionne. Pour la plupart des programmes Visual Basic, vous répéterez ce processus de nombreuses fois. En général, après avoir ajouté un nouveau code, vous exécutez le programme pour savoir si ce code s'exécute de la manière prévue. Dans la négative, vous devez le corriger. Ce processus est appelé *débogage* ; il sera examiné en détail dans une leçon ultérieure.

Le fait que votre programme navigue jusqu'à une page Web et en affiche le contenu grâce au simple ajout d'une seule ligne de code peut sembler magique. C'est toute la beauté de Visual Basic : tout le code nécessaire est intégré au contrôle **WebBrowser**. Vous économisez ainsi temps et efforts. Si vous deviez tout faire vous-même, il vous faudrait rédiger des centaines, voire des milliers, de lignes de code !

Dépannage

Si votre programme ne s'exécute pas ou n'affiche pas la page Web, vous pouvez vérifier certains éléments :

- Assurez-vous que vous êtes connecté à Internet. Ouvrez Internet Explorer et essayez de naviguer jusqu'à la page d'accueil du site Web de Microsoft. Si l'opération aboutit dans Internet Explorer, elle doit aboutir dans votre programme.
- Assurez-vous que vous avez saisi l'adresse (`http://www.microsoft.com/france`) correctement.
- Revenez à l'Étape 2 : créer une interface utilisateur et assurez-vous que vous avez bien disposé les contrôles appropriés sur le formulaire.
- Revenez à l'Étape 4 : ajouter du code Visual Basic et assurez-vous que vous avez tapé le code correctement.

Étapes suivantes

Félicitations ! Votre premier programme Visual Basic est terminé. Vous avez pu voir comment Visual Basic permet de développer rapidement et facilement des programmes puissants. Les leçons suivantes vous présenteront davantage de fonctionnalités du langage de programmation Visual Basic.

Leçon suivante : Introduction au langage de programmation Visual Basic

Voir aussi

Tâches

Étape 1 : créer un projet dans Visual Basic

Étape 2 : créer une interface utilisateur

Étape 3 : personnaliser l'apparence et le comportement

Étape 4 : ajouter du code Visual Basic

Autres ressources

Création de votre premier programme Visual Basic

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Introduction au langage de programmation Visual Basic

Microsoft Visual Basic 2005 Express est un moyen rapide et facile de créer des programmes pour Microsoft Windows. Même si vous débutez en programmation Windows, Visual Basic vous fournit un jeu complet d'outils qui simplifie le développement.

Mais qu'est-ce que Visual Basic ? Le terme « Visual » (visuel) fait référence à la méthode utilisée pour créer ce que l'utilisateur voit : l'*interface utilisateur graphique*, ou GUI (Graphical User Interface). "Basic" (de base) fait référence au langage de programmation BASIC (Beginners All-Purpose Symbolic Instruction Code), le langage le plus utilisé dans l'histoire de l'informatique. Vous pouvez créer des programmes utiles en apprenant simplement quelques-unes de ses fonctionnalités. Les liens suivants vous initieront à la programmation en Visual Basic ; chacun d'eux inclut des exemples et donne accès à des informations supplémentaires.

Concepts de programmation

Qu'est-ce exactement qu'un langage de programmation ? Les liens suivants vous fourniront des informations générales sur ce qu'est un langage et sur la manière dont il enregistre différents types d'informations.

Terme	Définition
Notions de base : fonctionnement de la programmation	Fonctionnement d'un langage de programmation et terminologie de base.
Représentation de mots, nombres et valeurs avec des variables	Stockage de valeurs et représentation d'informations par des variables, et utilisation des variables.
Mots et texte : utilisation de variables chaîne pour organiser des mots	Comment utiliser une variable **String** pour représenter des mots et un texte.
Tableaux : variables représentant plusieurs valeurs	Comment utiliser une variable **Array** pour représenter plusieurs valeurs du même type.
Arithmétique : création d'expressions avec des variables et opérateurs	Comment écrire un code qui exécute une opération arithmétique.
Comparaisons : utilisation d'expressions pour comparer des valeurs	Comment écrire un code qui compare des valeurs numériques.

Votre premier programme

Prêt pour un peu de vraie programmation ? Les liens suivants vous guideront pas à pas tout au long du processus de création d'un programme simple. Ils vous montreront comment rechercher d'éventuelles erreurs dans votre programme .

Terme	Définition
Charger l'ordinateur de certaines tâches : écriture de votre première procédure	Comment écrire un code qui spécifie à votre programme d'effectuer une action particulière.
Configuration de votre programme pour la répétition d'actions : exécution d'une boucle avec la boucle For...Next	Comment écrire un code qui répète des actions dans votre programme et qui compte les répétitions effectuées.

Deux possibilités pour votre programme : instruction If...Then	Comment écrire un code qui effectue des opérations différentes en réaction à des conditions différentes.
Opération à effectuer en cas de problème : gestion des erreurs	Comment écrire un code qui gère des erreurs dans vos programmes. Vous apprendrez également les différents types d'erreurs.

En savoir plus sur Visual Basic

Les liens suivants vous aideront à rentabiliser vos connaissances de la programmation et de Visual Basic 2005.

Terme	**Définition**

Présentation détaillée : explication des propriétés, des méthodes et des événements	Mode de fonctionnement des propriétés, méthodes et événements.

Présentation détaillée : types de données	Comment sont enregistrées les données à l'aide des différents types de variables.
Présentation détaillée : conversion d'un type de variable en un autre	Comment convertir des do

nnéesd'untypeennunautre;sontégalementexaminéscertainspiègescouraintslisàceprocessus.

Présentation détaillée : utilisation de Do...While et Do...Until pour poursuivre la répétition jusqu'à ce qu'une condition soit remplie	Comment utiliser les instructions **Do...While** et **Do...Until** pour répéter un code selon certaines conditions.

Présentation détaillée : utilisation de constructions Select Case pour choisir entre plusieurs options	Comment exécuter un code selon des conditions multiples où les choix sont nombreux.

Visite guidée de Visual Basic

Ce que vous pouvez faire de plus avec la langage de programmation Visual Basic 2005

Notions de base : fonctionnement de la programmation

Avant de vous lancer et de commencer à apprendre le *langage de programmation* Visual Basic, pour vous aider, examinons ce qu'est un langage de programmation, son fonctionnement et certains termes du domaine de la programmation. La meilleure approche consiste à commencer par les éléments fondamentaux.

Fonctionnement de la programmation

Seul, un ordinateur n'est pas très intelligent.

Un ordinateur est essentiellement un amas de commutateurs électroniques de très petite taille qui sont soit sous tension, soit hors tension. En configurant différentes combinaisons de ces commutateurs, vous pouvez faire faire des opérations à l'ordinateur ; par exemple, afficher quelque chose à l'écran ou produire un son. Voici donc en quoi la programmation consiste à son niveau le plus élémentaire : dire à un ordinateur ce qu'il doit faire.

Bien sûr, comprendre quelle combinaison de commutateurs fera faire à l'ordinateur l'opération que vous souhaitez constituerait une tâche décourageante ; c'est à ce stade que les langages de programmation interviennent.

Qu'est-ce qu'un langage de programmation ?

Les êtres humains s'expriment à l'aide d'un langage doté de nombreux mots. Les ordinateurs utilisent un langage simple qui se compose uniquement de 1 et de 0 ; 1 signifiant « activé » et 0 « désactivé ». Tenter de parler à un ordinateur dans son propre langage équivaudrait à parler à vos amis en morse ; c'est possible, mais à quoi bon ?

Un langage de programmation agit comme un traducteur entre vous et l'ordinateur. Plutôt que d'apprendre le langage natif de l'ordinateur, connu sous le nom de *langage machine*, vous pouvez utiliser un langage de programmation pour donner des instructions à l'ordinateur d'une manière plus facile à apprendre et à comprendre.

Un programme spécialisé, connu sous le nom de *compilateur*, récupère les instructions écrites en langage de programmation et les convertit en langage machine. Ainsi, en tant que programmeur Visual Basic, vous n'avez pas besoin de comprendre ce que fait l'ordinateur, ni comment il le fait ; vous devez simplement comprendre comment fonctionne le langage de programmation Visual Basic.

Au cœur du langage Visual Basic

À bien des égards, Visual Basic est très similaire à votre langage de tous les jours. Lorsque vous parlez ou écrivez, vous utilisez différents types de mots, tels que les noms ou les verbes, qui définissent leur mode opératoire. Visual Basic dispose, lui aussi, de différents types de mots, connus sous le nom d'*éléments de programmation*, qui définissent leur mode opératoire dans le cadre de l'écriture de programmes.

Les éléments de programmation présents dans Visual Basic sont notamment les *instructions*, les *déclarations*, les *fonctions*, les *opérateurs* et les *mots clés*. Au fil des leçons suivantes, vous en apprendrez davantage sur ces éléments et leur utilisation.

Écrit et parlé, un langage a également des règles, une *syntaxe* qui définit l'ordre des mots dans une phrase. Visual Basic a lui aussi une syntaxe ; elle peut sembler étrange de prime abord, mais elle est en fait très simple. Par exemple, pour déclarer "La vitesse maximale de ma voiture est de 55", vous écririez :

```
Car.Speed.Maximum = 55
```

Vous en apprendrez plus sur la syntaxe ultérieurement. Certains outils de Visual Basic, tels que la fonction *IntelliSense*, vous aident à employer la syntaxe appropriée lors de l'écriture de programmes.

Le langage que vous écrivez et parlez possède également une structure : par exemple, un livre comprend des chapitres avec des paragraphes qui contiennent des phrases. Les programmes écrits dans Visual Basic ont, eux aussi, une structure : les *modules* sont l'équivalent des chapitres, les *procédures* celui des paragraphes, et les *lignes de code* celui des phrases.

Étapes suivantes

Dans cette leçon, vous avez appris ce qu'est un langage de programmation et son fonctionnement. Dans la leçon suivante, vous commencerez à apprendre à utiliser le langage de programmation Visual Basic. Ne vous inquiétez pas ; vous parlerez le Visual Basic en un rien de temps !

Leçon suivante : Représentation de mots, nombres et valeurs avec des variables

Voir aussi

Tâches

Représentation de mots, nombres et valeurs avec des variables

Concepts

Présentation détaillée : explication des propriétés, des méthodes et des événements

Introduction au langage de programmation Visual Basic

Autres ressources

Création de votre premier programme Visual Basic

Guide de programmation Visual Basic

Représentation de mots, nombres et valeurs avec des variables

Les *variables* sont un concept important de la programmation informatique. Une variable est une lettre ou un nom qui peut stocker une valeur. Lorsque vous créez des programmes informatiques, vous pouvez utiliser des variables pour stocker des valeurs numériques, telles que la hauteur d'un bâtiment, ou des mots, tels que le nom d'une personne. Plus simplement, vous utilisez des variables pour représenter tout type d'informations nécessaire à votre programme.

Mais alors, pourquoi utiliser une variable alors qu'il suffirait d'utiliser l'information elle-même ? Comme leur nom l'indique, les variables peuvent modifier la valeur qu'elles représentent au fil de l'exécution du programme. Par exemple, vous pouvez écrire un programme pour suivre le nombre de bonbons en guimauve que contient un pot placé sur votre bureau. Sachant qu'un bonbon est fait pour être mangé, le nombre de ceux qui sont présents dans le pot va probablement évoluer au fil du temps. Plutôt que de réécrire votre programme à chaque fois que vous avez envie de sucre, vous pouvez représenter le nombre de bonbons par une variable qui évolue au fil du temps.

Stockage d'informations dans des variables

L'utilisation d'une variable comprend trois étapes :

1. **Déclarez la variable.** Informez le programme du nom et de la nature de la variable que vous souhaitez utiliser.
2. **Affectez la variable.** Attribuez à la variable une valeur qu'elle stockera.
3. **Utilisez la variable.** Récupérez la valeur contenue dans la valeur et utilisez-la dans votre programme.

Déclaration de variables

Lorsque vous déclarez une variable, vous devez décider de son nom et du *type de données* qui lui sera affecté.

Vous déclarez une variable à l'aide des mots clés **Dim** et **As** , comme illustré ci-dessous.

VB

```
Dim aNumber As Integer
```

Cette ligne de code informe le programme que vous souhaitez utiliser une variable nommée `aNumber` qui stocke des nombres entiers (type de données **Integer**).

Sachant que `aNumber` est de type **Integer**, il peut stocker uniquement des nombres entiers. Si vous souhaitiez stocker la valeur 42,5, par exemple, il vous aurait fallu utiliser le type de données **Double**. Et si vous souhaitiez stocker un mot, il vous aurait fallu utiliser le type de données **String**. Un autre de type de données qu'il faut mentionner à ce stade est **Boolean**. Il peut stocker une valeur `True` ou `False`.

Voici d'autres exemples de déclaration de variables.

VB

```
Dim aDouble As Double
Dim aName As String
Dim YesOrNo As Boolean
```

Pour plus d'informations sur les autres types de variables, consultez Présentation détaillée : types de données.

Affectation de variables

Vous affectez une valeur à votre variable avec le signe `=`, parfois appelé *opérateur d'assignation*, comme l'illustre l'exemple suivant.

VB

```
aNumber = 42
```

Cette ligne de code récupère la valeur `42` et la stocke dans la variable déclarée précédemment, nommée `aNumber`.

Déclaration et affectation de variables avec une valeur par défaut

Comme indiqué ci-dessus, vous pouvez déclarer une variable sur une ligne de code distincte, puis lui affecter ultérieurement une valeur sur une autre. Cette tactique peut générer une erreur si vous essayez d'utiliser la variable avant de lui affecter une valeur.

C'est pour cette raison qu'il est recommandé de déclarer et d'affecter des variables sur une seule et même ligne. Même si vous ignorez encore quelle valeur la variable va contenir, vous pouvez lui affecter une valeur par défaut. Le code pour déclarer et affecter les variables présentées précédemment aurait l'aspect suivant.

VB

```
Dim aDouble As Double = 0
Dim aName As String = "default string"
Dim YesOrNo As Boolean = True
```

En déclarant des variables et en affectant des valeurs par défaut sur une seule et même ligne, vous empêchez toute erreur potentielle et rien ne vous empêche ultérieurement d'affecter une valeur différente à la variable.

Essayez !

Dans cet exercice, vous allez écrire un programme court qui crée quatre variables, leur affecte des valeurs, puis affiche chaque valeur dans une fenêtre appelée *message*. Commençons par créer le projet dans lequel le code sera enregistré.

Pour créer le projet

1. Si ce n'est pas encore fait, ouvrez Visual Basic à partir du menu **Démarrer** de Windows.
2. Dans le menu **Fichier**, cliquez sur **Nouveau projet**.
3. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
4. Dans la zone **Nom**, tapez `Variables`, puis cliquez sur **OK**.

   Visual Basic crée les fichiers de votre programme et ouvre le **Concepteur de formulaires**.

Ensuite, vous allez créer les variables.

Pour créer des variables et afficher leurs valeurs

1. Double-cliquez sur le formulaire.

   L'éditeur de code s'ouvre à une section de code appelée `Form1_Load`. Cette section de code appelée *procédure* contient des instructions qui seront exécutées lors du premier chargement du formulaire en mémoire.

2. Dans la procédure `Form1_Load`, tapez le code suivant :

   VB

   ```
   Dim anInteger As Integer = 42
   Dim aSingle As Single = 39.345677653
   Dim aString As String = "I like candy"
   Dim aBoolean As Boolean = True
   ```

   Ce code déclare quatre variables : **Integer**, **Single**, **String** et **Boolean—**et leur assigne ensuite leur valeur par défaut..

   Conseil

   Lors de la saisie du code, vous avez pu remarquer après avoir tapé `As` l'affichage spontané d'une liste de mots sous le c urseur. Cette fonctionnalité s'appelle *Intellisense*. Elle permet de taper juste les premières lettres d'un mot jusqu'à ce qu e le mot soit sélectionné dans la liste. Une fois la sélection effectuée, appuyez sur la touche **TAB** pour insérer le mot.

   Remarque

Dès lors que vous souhaitez présenter le mot correspondant pour sa valeur propre dans un programme, vous devez le placer entre guillemets (""). Le programme interprète alors cette chaîne de caractères en tant que texte et non en tant q ue nom de variable. Lorsque vous affectez une valeur **True** ou **False** à une variable **Boolean**, vous ne placez pas le mot entre guillemets, car **True** et **False** sont des mots clés Visual Basic dotés d'une signification spéciale propre.

3. Sous le code que vous avez rédigé à l'étape précédente, tapez ce qui suit.

   VB

   ```
   MsgBox(anInteger)
   MsgBox(aSingle)
   MsgBox(aString)
   MsgBox(aBoolean)
   ```

   VB

   ```
   End
   ```

   Les quatre premières lignes de code demandent au programme d'afficher chaque valeur affectée à l'étape précédente dans une nouvelle fenêtre, à l'aide de la *fonction* **MsgBox**. La ligne finale demande au programme de s'arrêter après avoir exécuté cette procédure ; elle utilise l'*instruction* **End**.

4. Appuyez sur F5 pour exécuter le programme.

   Cliquez sur le bouton **OK** de chacune des fenêtres à mesure qu'elles s'affichent. Notez que les valeurs de chacune des variables s'affichent tour à tour. Ensuite, le programme s'arrête. Après avoir fini le programme, vous pouvez revenir en arrière et modifier les valeurs affectées dans le code, puis exécuter à nouveau l'application ; vous constaterez alors que les nouvelles valeurs s'affichent.

Étapes suivantes

Dans cette leçon, vous avez appris les notions de base des variables. Dans la leçon suivante, vous en apprendrez davantage sur les variables **String**.

Leçon suivante : Mots et texte : utilisation de variables chaîne pour organiser des mots

Voir aussi

Tâches

Mots et texte : utilisation de variables chaîne pour organiser des mots

Présentation détaillée : conversion d'un type de variable en un autre

Concepts

Notions de base : fonctionnement de la programmation

Présentation détaillée : types de données

Présentation détaillée : types de données

Les types de données en Visual Basic déterminent le type de valeurs ou de données qui peut être stocké dans une variable, ainsi que la manière dont ces données sont stockées. Pourquoi existe-t-il différents types de données ? Envisagez la question de la manière suivante : supposons trois variables, dont deux stockent des nombres tandis que la troisième stocke un nom : si les deux premières vous permettent d'appliquer une opération arithmétique, la troisième ne vous le permet pas. Affecter un type de données à une variable permet de déterminer plus facilement la manière dont la variable peut ou non être utilisée.

Remarque
Les types de données sont également utilisés dans d'autres éléments de programmation, notamment des constantes, des pr opriétés et des fonctions. Vous en saurez davantage sur les autres utilisations des types de données dans une leçon ultérieur e.

Types de données pour les nombres

La plupart des logiciels traitent des nombres d'une manière ou d'une autre. Sachant qu'il existe plusieurs façons d'exprimer des nombres, Visual Basic dispose de plusieurs types de données numériques pour traiter plus efficacement les nombres.

Le type de données numérique que vous utilisez le plus est le type **Integer**. Il sert à représenter un nombre entier (un nombre sans partie décimale). Lors du choix d'un type de données pour représenter des nombres entiers, utilisez de préférence le type de données **Long** si votre variable est amenée à stocker des nombres supérieurs à environ deux milliards ; dans la négative, un type **Integer** est plus efficace.

Tous les nombres ne sont pas des entiers ; par exemple, lorsque vous divisez deux nombres entiers, le résultat est souvent un nombre entier suivi d'une valeur décimale (9 divisé par 2 est égal à 4,5). Le type de données **Double** est utilisé pour représenter des nombres qui ont une partie décimale.

Remarque
Il existe d'autres types de données numériques, tels que **Decimal**, **Short**, **SByte** et **UInteger** ; ceux-ci sont généralement utili sés dans les programmes de très grande envergure où l'exploitation de la mémoire ou la vitesse pose problème. Pour le mo ment, les types de données numériques de base répondront à tous vos besoins. Pour en savoir plus sur les types de données avancés, consultez Types de données numériques.

Types de données pour le texte

La plupart des programmes traitent également du texte, qu'il s'agisse de présenter des informations à l'utilisateur ou de capturer le texte saisi par celui-ci. Le texte est généralement stocké dans le type de données **String** qui peut contenir une série de lettres, de nombres, d'espaces ou d'autres caractères. Un type **String** n'est pas limité en longueur : il peut s'agir d'une phrase, d'un paragraphe ou d'un caractère unique, voir de l'absence de caractère (une *chaîne NULL*).

Dans le cas d'une variable qui représentera systématiquement un seul caractère, il existe également le type de données **Char**. Si vous avez besoin qu'une variable unique ne stocke qu'un seul caractère, utilisez le type de données **Char** plutôt que le type **String**.

Autres types de données

En plus du texte et des nombres, les programmes sont parfois amenés à stocker d'autres types d'informations, tels qu'une valeur vraie ou fausse, une date, ou encore des données ayant une signification particulière pour le programme.

Pour les valeurs à représentation de type vrai/faux, oui/non, ou activé/désactivé, Visual Basic dispose du type de données **Boolean**. Une variable **Boolean** peut renfermer une valeur sur deux possibles : **True** ou **False**.

Bien que vous puissiez représenter des dates ou des heures sous la forme de nombres, le type de données **Date** facilite le calcul des dates ou des heures, notamment le nombre de jours jusqu'à votre anniversaire ou le nombre de minutes jusqu'au déjeuner.

Lorsque vous devez stocker plusieurs types de données dans une seule variable, vous pouvez utiliser un type de données *composite*. Les types de données composites comprennent des *tableaux*, des *structures* et des *classes*. Vous en saurez plus sur ces éléments dans les leçons ultérieures.

Enfin, le type des données que vous devez stocker peut être différent selon les circonstances. Le type de données **Object** vous

permet de déclarer une variable, puis d'en définir le type de données ultérieurement. Vous en saurez plus sur le type de données **Object** dans une leçon ultérieure.

Étapes suivantes

Dans la leçon suivante, intitulée « Mots et texte : utilisation de variables chaîne pour organiser des mots », vous en saurez plus sur la manière de travailler avec les variables chaîne pour constituer des phrases.

Leçon suivante : Mots et texte : utilisation de variables chaîne pour organiser des mots

Voir aussi

Tâches

Mots et texte : utilisation de variables chaîne pour organiser des mots

Représentation de mots, nombres et valeurs avec des variables

Concepts

Types de données en Visual Basic

Mots et texte : utilisation de variables chaîne pour organiser des mots

Dans cette leçon, vous apprendrez comment utiliser le type de données **String** pour représenter des mots et du texte.

Dans la leçon précédente, vous avez appris comment utiliser des variables pour stocker des données dans votre programme. Vous avez également appris que chaque variable doit être d'un type adapté aux données qu'elle stocke. Dans cette leçon, vous en saurez plus sur le type de données **String** , utilisé pour stocker du texte.

Qu'est-ce qu'une chaîne ?

Une *chaîne* (en anglais, string) est une série de caractères de texte, tels que lettres, nombres, caractères spéciaux et espaces. Les chaînes peuvent être expressions ou des phrases explicites, telles que « Rapide, le renard brun saute sur le chien paresseux », ou des combinaisons d'apparence inintelligible, telles que « @#fTWRE^3 35Gert ».

Les variables **String** sont créées comme les autres variables : elles sont d'abord déclarées puis se voient affecter une valeur, comme illustré ci-dessous.

VB

```
Dim aString As String = "This is a string"
```

Lors de l'affectation du texte proprement dit (également appelé *littéral de chaîne*) à une variable **String**, celui-ci doit figurer entre guillemets (""). Vous pouvez également utiliser le caractère = pour affecter une variable **String** à une autre variable **String**, comme l'illustré dans cet exemple.

VB

```
    Dim aString As String = "This is a string"
...
    Dim bString As String = ""
    bString = aString
```

Le code précédent attribue une valeur identique à `bString` et `aString` (`This is a string`).

Vous pouvez utiliser le caractère de perluète (`&`) pour combiner séquentiellement plusieurs chaînes au sein d'une nouvelle chaîne, comme indiqué ci-dessous.

VB

```
Dim aString As String = "Across the Wide"
Dim bString As String = "Missouri"
Dim cString As String = ""
cString = aString & bString
```

L'exemple précédent déclare trois variables **String** et affecte respectivement les valeurs "Across the Wide" et "Missouri" aux deux premières, puis affecte les valeurs combinées des deux premières à la troisième. D'après vous, qu'elle sera la valeur de `cString` ? Vous serez peut-être surpris d'apprendre que la valeur est `Across the WideMissouri`. En effet, il n'y a aucun espace ni à la fin de `aString`, ni au début de `bString`. Les deux chaînes sont simplement mises bout à bout. Si vous souhaitez intercaler un élément entre les deux chaînes, notamment un espace, vous devez le faire avec un *littéral de chaîne*, tel que `" "`, comme indiqué ci-dessous.

```
Dim aString As String = "Across the Wide"
Dim bString As String = "Missouri"
Dim cString As String = ""
cString = aString & " " & bString
```

Le texte contenu dans `cString` présente désormais l'aspect suivant : `Across the Wide Missouri`.

Essayez !

Pour joindre des chaînes

1. Dans le menu **Fichier**, cliquez sur **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet** :
   a. Dans le volet **Modèles**, cliquez sur **Application Windows**.
   b. Dans la zone **Nom**, tapez **Concatenation**.
   c. Cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
3. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
4. Dans la procédure événementielle **Form1.Load**, déclarez quatre variables chaîne et affectez les valeurs de chaîne indiquées ci-dessous :

   VB
   ```
   Dim aString As String = "Concatenating"
   Dim bString As String = "Without"
   Dim cString As String = "With"
   Dim dString As String = "Spaces"
   ```
5. Ajoutez le code suivant pour concaténer les chaînes et afficher les résultats :

   VB
   ```
       ' Displays "ConcatenatingWithoutSpaces".
       MsgBox(aString & bString & dString)

   ...     ' Displays "Concatenating With Spaces".
       MsgBox(aString & " " & cString & " " & dString)
   ```

   Le texte affiché dans le message est le résultat de la jointure des variables chaîne qui ont été affectées dans une étape précédente. Dans la première zone, les chaînes sont jointes sans espace. Dans la seconde, des espaces sont insérés explicitement entre les chaînes.

Étapes suivantes

Dans cette leçon, vous avez appris comment déclarer et affecter des chaînes (string), et comment les joindre avec l'opérateur de concaténation &. Dans la leçon suivante, « Tableaux : variables représentant plusieurs valeurs », vous apprendrez comment créer une variable pour enregistrer des groupes d'éléments similaires.

Leçon suivante : Tableaux : variables représentant plusieurs valeurs

Voir aussi

Tâches

Tableaux : variables représentant plusieurs valeurs

Représentation de mots, nombres et valeurs avec des variables

Concepts

Introduction au langage de programmation Visual Basic

Tableaux : variables représentant plusieurs valeurs

Dans cette leçon, vous apprendrez comment utiliser des *tableaux* pour enregistrer des groupes de valeurs.

Comme vous l'avez appris dans les leçons précédentes, les variables sont utilisées pour stocker différents types de données que votre programme pourra utiliser. Un autre type de variable, appelé *tableau*, constitue un moyen pratique d'enregistrer plusieurs valeurs de même type.

Par exemple, vous écrivez un programme pour une équipe de base-ball et vous souhaitez enregistrer les noms de tous les joueurs présents sur le terrain. Vous pouvez créer neuf variables chaîne distinctes, une par joueur, ou déclarer une variable tableau qui ressemble au code ci-dessous.

VB

```
Dim players() As String
```

Vous déclarez une variable tableau en plaçant des parenthèses après le nom de variable. Si vous savez déjà combien de valeurs vous devez stocker, vous pouvez spécifier également la taille du tableau dans la déclaration de la manière suivante :

VB

```
Dim players(8) As String
```

Vous pouvez trouver curieux que la taille du tableau soit de `8`, alors qu'une équipe de base-ball compte `9` joueurs. Il faut savoir qu'un tableau est composé de plusieurs valeurs, ou *éléments*, et que ceux-ci commencent au numéro 0 et se terminent au numéro spécifié dans la déclaration. Dans ce cas, le tableau contient les éléments 0 à 8, pour un total de neuf éléments.

Affectation de valeurs aux tableaux

Comme avec d'autres types de valeurs, vous devez affecter des valeurs aux tableaux. Pour ce faire, vous vous référez au numéro d'élément dans le cadre de l'affectation, comme illustré ci-dessous.

VB

```
players(0) = "John"
players(3) = "Bart"
```

Dans le code ci-dessus, la valeur `John` est affectée au premier élément du tableau (élément 0) et la valeur `Brett` au quatrième (élément 3). Il n'est pas nécessaire d'affecter les éléments du tableau dans l'ordre. Tout élément non affecté se verra attribuer une valeur par défaut ; dans le cas présent, une chaîne vide.

Comme avec d'autres types de valeurs, vous pouvez déclarer et assigner des valeurs à un tableau sur une seule ligne, de la manière suivante.

VB

```
Dim players() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Dans ce cas, des accolades indiquent une liste de valeurs. Les valeurs sont affectées aux éléments dans l'ordre spécifié. Remarquez que la taille du tableau n'est pas spécifiée. Elle est déterminée par le nombre des éléments que vous énumérez.

Récupération de valeurs dans des tableaux

Tout comme vous utilisez des nombres pour spécifier la position d'un élément dans un tableau, vous utilisez le numéro d'élément pour spécifier la valeur que vous souhaitez récupérer.

VB

```
Dim AtBat As String
AtBat = players(3)
```

Le code ci-dessus récupère le quatrième élément du tableau et l'affecte à la variable chaîne `AtBat`.

Essayez !

Pour enregistrer des valeurs dans un tableau

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `MyFirstArray`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle **TextBox** dans le formulaire.
5. À partir de la **Boîte à outils**, faites glisser un contrôle **Button** jusqu'au formulaire.
6. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.
7. Dans la procédure de l'événement **Button1_Click**, ajoutez le code suivant :

   VB

   ```
   Dim players() As String = {"Dan", "Fred", "Bart", "Carlos", _
     "Ty", "Juan", "Jay", "Sam", "Pedro"}
   Dim i As Integer = CInt(Textbox1.Text)
   MsgBox(players(i) & " is on first base.")
   ```

   Remarquez que le code ci-dessus utilise la fonction **CInt** pour convertir la valeur **String** (`TextBox1.Text`) en **Integer** (`i`). Pour en savoir davantage sur les conversions, consultez Présentation détaillée : conversion d'un type de variable en un autre.
8. Appuyez sur F5 pour exécuter le programme.
9. Entrez un nombre entre 0 et 8 dans la zone de texte et cliquez sur le bouton. Le nom qui correspond à cet élément s'affiche dans un message.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser des tableaux pour stocker et récupérer des groupes de valeurs semblables.

Dans la leçon suivante, vous apprendrez comment utiliser des opérateurs arithmétiques pour créer des expressions.

Leçon suivante : Arithmétique : création d'expressions avec des variables et opérateurs

Voir aussi

Tâches

Arithmétique : création d'expressions avec des variables et opérateurs

Mots et texte : utilisation de variables chaîne pour organiser des mots

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Arithmétique : création d'expressions avec des variables et opérateurs

Dans cette leçon, vous apprendrez comment créer des expressions pour exécuter des opérations arithmétiques puis retourner des valeurs.

Une *expression* est un segment de code qui exécute une opération arithmétique puis retourne une valeur. Par exemple, une expression d'addition simple est présentée ci-dessous.

5 + 4

L'expression `5 + 4` retourne la valeur `9` lorsqu'elle est évaluée. Elle se compose de deux parties : les *opérandes* (`5` et `4`), qui sont les valeurs auxquelles l'opération s'applique, et l'*opérateur* (`+`), qui spécifie l'opération à effectuer.

Utilisation des valeurs retournées par les expressions

Pour qu'une expression soit utile, vous devez exploiter la valeur qu'elle retourne. La méthode la plus répandue consiste à l'affecter à une variable, comme illustré ci-dessous.

VB

```
Dim anInteger As Integer = 5 + 4
```

Cet exemple déclare une nouvelle variable **Integer**, appelée `anInteger`, et lui affecte la valeur retournée par `5 + 4` .

Opérateurs arithmétiques

Les expressions sont généralement utilisées pour appliquer des opérations arithmétiques à des variables : addition, soustraction, multiplication ou division. Le tableau suivant décrit les opérateurs arithmétiques les plus courants.

Opérateur	Description	Exemple
+ (addition)	Retourne la somme de deux opérandes.	5 + 4
- (soustraction)	Retourne la différence de deux opérandes.	5 - 4
* (multiplication)	Retourne le produit de deux opérandes.	5 * 4
/ (division)	Retourne le quotient de deux opérandes.	5 / 4

Le type de variable que vous utilisez lors de l'exécution d'une opération arithmétique peut affecter le résultat. Diviser deux nombres entraîne souvent le retour d'une valeur décimale. Par exemple, lorsque vous divisez 3 par 2, le résultat est 1,5. Si vous avez affecté la valeur de retour de cette expression à une variable **Integer**, cette valeur est arrondie au nombre entier le plus proche, 2. Lorsque vous effectuez une division, vous devez utiliser une variable **Double** pour stocker la valeur de retour.

Remarque

Vous pouvez également convertir une variable d'un type de données en un autre, en utilisant les fonctions de conversion de Visual Basic. Pour plus d'informations, consultez Présentation détaillée : conversion d'un type de variable en un autre.

Essayez !

Pour additionner deux nombres

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Arithmetic`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser deux contrôles **Textbox** dans le formulaire.

5. À partir de la **Boîte à outils**, faites glisser un contrôle **Button** jusqu'au formulaire.
6. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.
7. Dans la procédure de l'événement **Button1_Click**, tapez le code suivant :

   VB

   ```
   Dim A As Double = Textbox1.Text
   Dim B As Double = Textbox2.Text

   MsgBox(A + B)
   MsgBox(A - B)
   MsgBox(A * B)
   MsgBox(A / B)
   ```

   Les deux premières lignes déclarent les variables `A` et `B` qui stockent les valeurs numériques utilisées dans ce programme, et affectent les valeurs des deux contrôles **TextBox** (leur texte) aux variables `A` et `B`.

   Les quatre dernières lignes créent des expressions avec les deux variables et chacun des opérateurs arithmétiques de base, puis affichent les résultats de ces expressions dans un message.
8. Appuyez sur F5 pour exécuter l'application.
9. Tapez un nombre dans chacune des zones de texte, puis cliquez sur **Button1**.

Remarque
Si vous tapez tout autre caractère dans les zones de texte, une erreur se produit.

   Les expressions sont créées au moyen des deux nombres que vous entrez et de chacun des quatre opérateurs arithmétiques de base (addition, soustraction, multiplication et division). Le résultat de chaque expression s'affiche dans un message.

Étapes suivantes

Dans cette rubrique, vous avez appris à créer et à utiliser des expressions. Vous avez également appris ce que sont les opérandes et les opérateurs, ainsi que la manière de construire une expression. À ce stade, vous pouvez soit passer à la leçon suivante, Comparaisons : utilisation d'expressions pour comparer des valeurs, soit examiner plus en détail la conversion de variables en différents types dans Présentation détaillée : conversion d'un type de variable en un autre. Si vous choisissez « Présentation détaillée », vous optez pour la seconde possibilité, puis passez à la leçon suivante.

Voir aussi

Tâches

Présentation détaillée : conversion d'un type de variable en un autre

Concepts

Opérateurs arithmétiques en Visual Basic

Présentation détaillée : conversion d'un type de variable en un autre

Comme vous avez pu le constater, les variables sont de différents types. Le type détermine le genre de données qu'une variable peut contenir. Une variable **Integer** peut contenir uniquement des données numériques sans décimale. Une variable **String** peut contenir uniquement du texte.

Que se passe-t-il lorsque vous souhaitez afficher des données de type **Integer** dans un contrôle TextBox qui requiert un type **String** ? Le type de ces données doit être converti. Dans cette rubrique, vous examinerez la manière de convertir des données d'un type en un autre, et apprendrez quelques techniques de conversion de données, ainsi que certains des pièges communs en la matière.

Conversion de variables en texte

Chaque variable dans Visual Basic peut être convertie en texte en utilisant une fonction spéciale appelée **CStr** (version abréviée de **C**onvert to **Str**ing). Cette fonction, comme son nom l'indique, retourne les données représentées par la variable sous la forme d'un type **String**. La procédure suivante montre un exemple simple de la conversion d'un type **Integer** en texte.

Essayez !

Pour convertir une variable en texte

1. Dans le menu **Fichier**, cliquez sur **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Conversion`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans le gestionnaire d'événements **Form1_Load**, tapez le code suivant :

   VB
   ```
   Dim anInteger As Integer = 54
   MsgBox(CStr(anInteger))
   ```

   Ce code déclare une variable Integer nommée `anInteger`, lui affecte la valeur 54, convertit cette valeur en texte et l'affiche dans un message en appelant la fonction **CStr**.
6. Appuyez sur F5 pour générer et exécuter votre application. Un message s'affiche et présente `54`.

   Essayons autre chose pour nous amuser. Dans l'éditeur de code, changez la ligne du code `MsgBox(CStr(anInteger))` pour lire `MsgBox(anInteger)`, puis appuyez sur F5 pour l'exécuter. Que se passe-t-il ? Le programme se comporte exactement comme auparavant. Visual Basic est suffisamment intelligent pour savoir que vous souhaitez, en réalité, convertir le type **Integer** en un texte à afficher dans le message. Toutefois, ce comportement ne suffit pas à tous les cas ; de nombreux types de variable ne peuvent pas être convertis automatiquement. Par conséquent, il est recommandé d'utiliser systématiquement la fonction **CStr**, même lorsqu'une variable est convertie automatiquement en texte.

En plus de convertir les variables **Integer** en texte, la fonction **CStr** peut s'appliquer à tout type de données numériques, tel que **Double** ou **Long**. Elle peut également s'utiliser pour convertir les types de données **Date** et **Boolean** en texte. Pour plus d'informations sur les types de données, consultez Présentation détaillée : types de données.

Conversion entre types de données numériques

Comme vous l'avez appris dans cette leçon d'arithmétique, le résultat d'un calcul ne peut pas être exprimé sous la forme d'un type **Integer**. Tout comme il dispose d'une fonction de conversion des nombres en texte, Visual Basic dispose également de fonctions pour convertir des variables d'un type de données numérique en un autre. Par exemple, vous pouvez utiliser la fonction **CDbl** (**C**onvert to **D**ou**bl**e) dans une opération arithmétique pour retourner un nombre décimal lorsque vous travaillez avec des variables de type **Integer**. La procédure suivante illustre la manière d'utiliser la fonction **CDbl** pour la division de deux entiers.

Essayez !

Pour convertir des types de données numériques

1. Dans l'éditeur de code, supprimez le code qui vous avez saisi dans la procédure précédente, puis tapez les éléments suivants :

   VB

   ```
   Dim A As Integer = 1
   Dim B As Integer = 2
   MsgBox(CDbl(A / B))
   ```

   Ce code déclare deux variables **Integer** (`A` et `B`), leur affecte les valeurs `1` et `2`, puis convertit le résultat de la division (`A / B`) à l'aide de la fonction **CDbl**, et l'affiche dans un message.

2. Appuyez sur F5 pour générer et exécuter votre application. Un message s'affiche et présente `0.5`.

Visual Basic dispose également de fonctions destinées aux autres types de variables numériques. Par exemple, si vous ajoutez deux variables du type **Double** et souhaitez arrondir le résultat au nombre entier le plus proche, utilisez la fonction **CInt**. Les autres fonctions de conversion numérique sont notamment **CByte**, **CDec**, **CLng** et **CShort**. Pour obtenir une liste de toutes les fonctions de conversion disponibles dans Visual Basic, consultez Fonctions de conversion de types de données.

Étapes suivantes

Dans cette leçon, vous avez appris comment convertir des variables numériques en texte, et comment interconvertir différents types de variables numériques. Dans la leçon suivante, Comparaisons : utilisation d'expressions pour comparer des valeurs, vous apprendrez comment évaluer des expressions.

Voir aussi

Tâches

Arithmétique : création d'expressions avec des variables et opérateurs

Concepts

Présentation détaillée : types de données

Autres ressources

Conversions de type en Visual Basic

Comparaisons : utilisation d'expressions pour comparer des valeurs

Dans cette leçon, vous apprendrez comment utiliser des *opérateurs de comparaison* pour créer des expressions qui comparent des valeurs.

Dans la dernière leçon, vous avez appris comment utiliser des opérateurs arithmétiques pour créer des expressions numériques et retourner des valeurs numériques. Un autre genre d'opérateurs, les *opérateurs de comparaison*, peut être utilisé pour comparer des valeurs numériques et retourner des valeurs de type **Boolean** (**True** ou **False**).

Les opérateurs de comparaison sont fréquemment utilisés pour comparer des valeurs et prendre des décisions en fonction de la comparaison réalisée. Les décisions prises dans votre programme sont examinées en détail dans Deux possibilités pour votre programme : instruction If...Then.

Le tableau suivant récapitule les opérateurs de comparaison :

Opérateur	Description	Exemples
= (égal)	Retourne **True** si le nombre placé à gauche est égal au nombre placé à droite.	5 = 4 (false) 4 = 5 (false) 4 = 4 (true)
<> (différent de)	Retourne **True** si le nombre placé à gauche est différent du nombre placé à droite.	5 < > 4 (True) 4 < > 5 (True) 4 <> 4 (false)
> (supérieur à)	Retourne **True** si le nombre placé à gauche est supérieur au nombre placé à droite.	5 > 4 (True) 4 > 5 (False) 4 > 4 (False)
< (inférieur à)	Retourne **True** si le nombre placé à gauche est inférieur au nombre placé à droite.	5 < 4 (False) 4 < 5 (True) 4 < 4 (False)
>= (supérieur ou égal à)	Retourne **True** si le nombre placé à gauche est supérieur ou égal au nombre placé à droite.	5 >= 4 (True) 4 >= 5 (False) 4 >= 4 (True)
<= (inférieur ou égal à)	Retourne **True** si le nombre placé à gauche est inférieur ou égal au nombre placé à droite.	5 <= 4 (False) 4 <= 5 (True) 4 <= 4 (True)

Essayez !

Pour comparer des expressions

1. Dans le menu **Fichier**, cliquez sur **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.

3. Dans la zone **Nom**, tapez `Comparison`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.

4. À partir de la **Boîte à outils**, faites glisser deux contrôles **Textbox** dans le formulaire.

5. À partir de la **Boîte à outils**, faites glisser un contrôle **Button** jusqu'au formulaire.

6. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.

7. Dans le gestionnaire d'événements **Button1_Click**, tapez le code suivant :

   VB

   ```
   Dim A As Double = CDbl(Textbox1.Text)
   Dim B As Double = CDbl(Textbox2.Text)
   MsgBox(A > B)
   MsgBox(A < B)
   MsgBox(A = B)
   ```

   Les deux premières lignes déclarent les variables `A` et `B` qui stockent les valeurs numériques utilisées dans ce programme ; elles utilisent l'instruction `CDbl` pour convertir le texte de `Textbox1` et `Textbox2` en valeurs numériques. Enfin, les trois dernières lignes créent des expressions pour comparer les deux variables au moyen de trois opérateurs de comparaison de base, et afficher les résultats de ces expressions dans trois messages.

8. Appuyez sur F5 pour exécuter l'application.

9. Tapez un nombre dans chacune des zones de texte, puis cliquez sur **Button1.**

   Le premier message affiche la valeur **True** si `A` (le nombre que vous avez entré dans la première zone de texte) est supérieur à `B` (le nombre que vous avez entré dans la seconde zone de texte) ; dans la négative, il affiche la valeur **False**. Le deuxième message affiche **True** si `A` est inférieur à `B`, et le troisième message affiche **True** si les deux nombres sont identiques.

   Entrez différents nombres dans les zones de texte et examinez les différents résultats.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser des opérateurs de comparaison pour comparer des valeurs numériques. Dans la leçon suivante, vous apprendrez comment créer et appeler une *procédure* (code qui exécute une action).

Leçon suivante : Charger l'ordinateur de certaines tâches : écriture de votre première procédure

Voir aussi

Tâches

Arithmétique : création d'expressions avec des variables et opérateurs
Présentation détaillée : conversion d'un type de variable en un autre
Deux possibilités pour votre programme : instruction If...Then

Concepts

Opérateurs de comparaison en Visual Basic

Charger l'ordinateur de certaines tâches : écriture de votre première procédure

Dans cette leçon, vous apprendrez comment créer une *procédure*, un bloc de code autonome qui peut être exécuté à partir d'autres blocs de code, puis des *paramètres* pour vos procédures.

Une procédure est simplement un code qui demande au programme d'exécuter une action. Sans vous en rendre compte, vous avez déjà utilisé des procédures dans les leçons précédentes. Par exemple, la fonction **MsgBox** intègre une procédure qui affiche une boîte de dialogue.

Si Visual Basic intègre de nombreuses procédures pour exécuter des actions communes, il y a toujours des cas où vous souhaitez que votre programme exécute une action qu'une procédure intégrée ne peut pas gérer. Par exemple, la fonction **MsgBox** ne peut afficher aucune boîte de dialogue dotée d'une image. Pour effectuer cette tâche, vous devez écrire votre propre procédure.

Qu'est-ce qu'une procédure ?

Une procédure est un bloc de code autonome qui peut être exécuté à partir d'autres blocs de code. En général, les procédures contiennent chacune le code nécessaire pour accomplir une tâche. Par exemple, vous pouvez disposer d'une procédure appelée **PlaySound** qui contient le code nécessaire à la lecture d'un fichier .wav. Même si vous pouvez écrire le code pour lire un son chaque fois que votre programme doit en émettre un, il est plus rationnel de créer une procédure unique que vous pouvez appeler depuis n'importe quel emplacement dans votre programme.

Une procédure s'*exécute* en *l'appelant* dans un code. Par exemple, pour exécuter la procédure **PlaySound**, vous devez simplement ajouter une ligne de code à votre programme avec le nom de votre procédure, comme indiqué ci-dessous.

```
PlaySound
```

Et c'est tout ! Lorsque le programme atteint cette ligne, il accède à la procédure **PlaySound** et exécute le code qu'il trouve à son emplacement. Le programme revient ensuite à la ligne qui suit celle de l'appel à **PlaySound**.

Vous pouvez appeler autant de procédure que vous le souhaitez. Les procédures s'exécutent dans l'ordre d'appel. Par exemple, si vous disposez également d'une procédure appelée **DisplayResults**, l'exécuter après avoir exécuté la procédure **PlaySounds**, appelle ces procédures conformément au code ci-dessous.

```
PlaySounds

DisplayResults
```

Fonctions et sous-routines

Il existe deux types de procédures : les *fonctions* et les *sous-routines* (parfois appelées *sub*). Une fonction retourne une valeur à la procédure qui l'a appelée, tandis qu'une sous-routine se contente d'exécuter un code. Les sous-routines sont appelées lorsqu'une ligne de code qui contient leurs noms est ajoutée au programme, comme dans l'exemple suivant.

```
DisplayResults
```

Les fonctions sont différentes, car non seulement elles exécutent le code, mais elles retournent également une valeur. Par exemple, imaginez une fonction appelée **GetDayOfWeek** qui retourne un **Integer** qui indique le jour de la semaine. Vous appelez cette fonction en déclarant d'abord une variable pour stocker la valeur retournée, puis en assignant la valeur retournée à la variable pour l'utiliser ultérieurement, comme illustré ci-dessous.

```
Dim Today As Integer

Today = GetDayOfWeek
```

Dans cet exemple, la valeur retournée par la fonction est copiée dans la variable nommée `Today`, puis enregistrée à des fins d'utilisation ultérieure.

Écriture de procédures

Vous écrivez des procédures en écrivant en premier lieu une *déclaration de procédure*. Une déclaration de procédure a plusieurs objectifs : elle déclare si la procédure est une fonction ou une sous-routine, nomme la procédure, et détaille tous les paramètres éventuels de la procédure (les paramètres seront examinés en détail dans cette leçon). L'exemple suivant est un exemple de déclaration de procédure simple.

VB

```
Sub MyFirstSub()
```

```
End Sub
```

Le mot clé `Sub` explique au programme que cette procédure est une sous-routine et ne retournera aucune valeur. Le nom de la sous-routine (`MyFirstSub`) vient ensuite. Les parenthèses vides indiquent l'absence de *paramètres* dans cette procédure. Enfin, le mot clé **End Sub** indique la fin de la sous-routine. Tout le code qui sera exécuté par cette sous-routine se place entre ces deux lignes.

Déclarer des fonctions est similaire, à ceci près que vous devez spécifier le type de la valeur retournée (par exemple, **Integer** ou **String**). Par exemple, une fonction qui a retourné un **Integer** peut ressembler à ce qui suit.

VB

```
Function MyFirstFunction() As Integer
End Function
```

Les mots clés `As Integer` indiquent que cette fonction retourne une valeur de type **Integer** . Pour retourner une valeur à partir d'une fonction, utilisez le mot clé **Return**, comme illustré dans l'exemple suivant.

VB

```
Function GetTheNumberOne() As Integer
    Return 1
End Function
```

Cette procédure retourne le nombre 1.

Essayez !

Pour créer des procédures

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `MyFirstProcedure`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans l'éditeur de code, repérez la ligne `End Class`. Il s'agit de la fin de la section de code qui constitue votre formulaire. Juste avant cette ligne, ajoutez la procédure suivante :

   VB

   ```
   Function GetTime() As String
     Return CStr(Now)
   End Function
   ```

   Cette fonction utilise la procédure intégrée **Now** pour obtenir l'heure actuelle, puis utilise la fonction **CStr** pour convertir la valeur retournée par **Now** à un format explicite de type **String**. Enfin, cette valeur **String** est retournée en tant que résultat de la fonction.
6. Au-dessus de la fonction ajoutée à l'étape précédente, ajoutez le **Sub** suivant.

   VB

   ```
   Sub DisplayTime()
     MsgBox(GetTime)
   End Sub
   ```

Cette sous-routine appelle la fonction `GetTime` et affiche le résultat qu'elle retourne dans un message.

7. Enfin, ajoutez une ligne au gestionnaire d'événements **Form1_Load** qui appelle la sous-routine **DisplayTime**, comme illustré ci-dessous.

   VB

   ```
   DisplayTime()
   ```

8. Appuyez sur F5 pour exécuter le programme.

   Lorsque le programme démarre, la procédure événementielle `Form1_Load` s'exécute. Cette procédure appelle la sous-routine **DisplayTime** ; aussi, l'exécution du programme passe directement à la sous-procédure **DisplayTime**. Cette sous-routine appelle à son tour la fonction **GetTime** ; aussi l'exécution du programme passe directement à la fonction **GetTime**. Cette fonction retourne un élément **String** qui représente l'heure à la sous-procédure **DisplayTime**, qui affiche alors cette chaîne dans un message. Une fois l'exécution de la sous-routine terminée, le programme reprend normalement et affiche le formulaire.

Paramètres des fonctions et des sous-routines

Vous devrez parfois fournir des informations supplémentaires à vos procédures. Par exemple, vous pouvez souhaiter jouer un son parmi plusieurs dans la procédure **PlaySound**. Vous pouvez fournir des informations sur l'identité du son à jouer en utilisant des paramètres.

Les paramètres sont très similaires aux variables. Ils disposent d'un type et d'un nom, et ils stockent des informations, comme les variables. Ils s'utilisent comme des variables dans une procédure. Les deux principales différences entre les paramètres et les variables sont les suivantes :

- Les paramètres sont déclarés dans la déclaration de procédure, et non individuellement sur des lignes de code distinctes.
- Les paramètres ne peuvent être utilisés que dans la procédure dans laquelle ils sont déclarés.

Les paramètres sont déclarés dans la déclaration de procédure, entre les parenthèses qui suivent le nom de procédure. Le mot clé **As** est utilisé pour déclarer le type, et chaque paramètre est généralement précédé par le mot clé **ByVal**. Ce mot clé est ajouté automatiquement par Visual Basic si vous ne l'ajoutez pas. Son fonctionnement est assez avancé et dépasse le cadre de cette leçon.

Un exemple de sous-routine avec ses paramètres est proposé ci-dessous.

VB

```
Sub PlaySound(ByVal SoundFile As String, ByVal Volume As Integer)
  My.Computer.Audio.Play(SoundFile, Volume)
End Sub
```

Vous appelez ensuite la sous-routine avec les valeurs destinées aux paramètres, comme illustré ci-dessous.

VB

```
PlaySound("Startup.wav", 1)
```

Vous pouvez également déclarer des paramètres pour les fonctions, exactement comme vous le faites pour les sous-routines.

Essayez !

Pour créer une fonction dotée de paramètres

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `parameters`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.

4. À partir de la **Boîte à outils**, faites glisser deux contrôles **Textbox** dans le formulaire.
5. À partir de la **Boîte à outils**, faites glisser un contrôle **Button** jusqu'au formulaire.
6. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.
7. Immédiatement après la ligne **End Sub** du gestionnaire d'événements `Button1_Click`, ajoutez la procédure suivante :

   VB

   ```
   Function AddTwoNumbers(ByVal N1 As Integer, ByVal N2 As Integer) _
     As Integer
     Return N1 + N2
   End Function
   ```

8. Dans la procédure `Button1_Click`, ajoutez le code suivant :

   VB

   ```
   Dim aNumber As Integer = CInt(Textbox1.Text)
   Dim bNumber As Integer = CInt(Textbox2.Text)
   MsgBox(AddTwoNumbers(aNumber, bNumber))
   ```

   Ce code déclare deux entiers et convertit le texte contenu dans les deux zones de texte en valeurs entières. Il passe alors ces valeurs à la fonction **AddTwoNumbers** et affiche la valeur retournée dans un message.
9. Appuyez sur F5 pour exécuter le programme.
10. Tapez une valeur numérique dans chaque zone de texte, puis cliquez sur le bouton. Les deux nombres sont ajoutés et le résultat s'affiche dans un message.

Étapes suivantes

Dans cette leçon, vous avez appris la différence entre des fonctions et des sous-routines, ainsi que la manière de créer chacune de ces deux procédures. Vous avez également appris comment appeler des procédures et comment créer des procédures possédant des paramètres.

Dans la leçon suivante, vous apprendrez comment utiliser l'instruction **For...Next** pour répéter des actions.

Leçon suivante :
Configuration de votre programme pour la répétition d'actions : exécution d'une boucle avec la boucle For...Next

Voir aussi

Tâches

Comparaisons : utilisation d'expressions pour comparer des valeurs

Concepts

Procédures dans Visual Basic

Configuration de votre programme pour la répétition d'actions : exécution d'une boucle avec la boucle For...Next

Dans cette leçon, vous apprendrez comment utiliser l'instruction **For...Next** pour répéter des actions dans votre programme, et pour compter combien de fois ces actions ont été effectuées.

Lorsque vous écrivez un programme, vous devez fréquemment répéter des actions. Par exemple, supposons que vous écriviez une méthode qui affiche une série de nombres à l'écran et que vous souhaitiez répéter la ligne de code qui affiche les nombres autant de fois que nécessaire.

La boucle **For...Next** vous permet de spécifier un nombre, puis de répéter un code qu'elle contient un nombre de fois spécifié. L'exemple suivant affiche la manière dont une boucle **For...Next** se présente dans un code.

VB

```
Dim i As Integer = 0
For i = 1 To 10
  DisplayNumber(i)
Next
```

La boucle **For...Next** commence par une *variable de compteur*, `i`. La boucle utilise cette variable pour compter le nombre de fois qu'elle s'est exécutée. La ligne suivante (`For i = 1 to 10`) indique au programme le nombre de fois qu'il doit répéter la boucle, ainsi que les valeurs que `i` va recevoir.

Lorsque le code entre dans la boucle **For...Next**, il commence par `i` qui contient la première valeur (dans le cas présent, 1). Le programme exécute ensuite les lignes de code placées entre la ligne `For` et la ligne `Next` ; dans le cas présent, il appelle la méthode `DisplayNumber` avec un paramètre de `i` (ici encore, 1).

Lorsque le programme atteint la ligne `Next`, `1` est ajouté à `i` et l'exécution du programme effectue un nouveau saut en arrière jusqu'à la ligne `For`. Ce processus se répète jusqu'à ce que la valeur de `i` soit supérieure au deuxième nombre présent sur la ligne `For` ; dans le cas présent, 10. Lorsque cette condition est remplie, le programme poursuit son exécution avec le code placé après la ligne `Next`.

Essayez !

Pour utiliser une instruction For...Next

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `ForNext`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Depuis la **Boîte à outils**, faites glisser un contrôle **TextBox** et un contrôle **Button** dans le formulaire.
5. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.
6. Dans le gestionnaire d'événements **Button1_Click**, tapez le code suivant :

   VB

   ```
   Dim i As Integer = 0
   Dim NumberOfRepetitions As Integer = CInt(Textbox1.Text)
   For i = 1 To NumberOfRepetitions
     MsgBox("This line has been repeated " & i & " times")
   Next
   ```
7. Appuyez sur F5 pour exécuter le programme.
8. Dans la zone de texte, tapez un nombre, puis cliquez sur le bouton.

Un message s'affiche autant de fois que vous l'avez spécifié dans la zone de texte.

Étapes suivantes

Dans cette rubrique, vous avez appris comment utiliser la boucle **For...Next** pour répéter un code un nombre de fois spécifié. À ce stade, vous pouvez soit passer à la leçon suivante de la série, Deux possibilités pour votre programme : instruction If...Then, soit examiner un autre type de boucle dans
Présentation détaillée : utilisation de Do...While et Do...Until pour poursuivre la répétition jusqu'à ce qu'une condition soit remplie.

Voir aussi

Tâches

Charger l'ordinateur de certaines tâches : écriture de votre première procédure

Référence

For...Next, instruction (Visual Basic)

Concepts

Structures de décision

Présentation détaillée : utilisation de Do...While et Do...Until pour poursuivre la répétition jusqu'à ce qu'une condition soit remplie

Dans cette leçon, vous apprendrez comment utiliser les instructions **Do...While** et **Do...Until** pour répéter un code selon certaines conditions.

Dans la leçon précédente, vous avez appris comment utiliser l'instruction **For...Next** pour appliquer une boucle à un bloc de code un nombre de fois spécifié. Mais qu'en est-il si le nombre de fois que le code doit être répété diffère en fonction de conditions données ? Les instructions **Do...While** et **Do...Until** vous permettent de répéter un bloc de code tant qu'une certaine condition a la valeur **True**, ou jusqu'à ce qu'une certaine condition affiche la valeur **True**.

Par exemple, supposez qu'un de vos programmes ajoute une série de nombres. Toutefois, vous souhaitez que la somme de ces nombres ne soit jamais supérieure à 100. Pour ce faire, vous pouvez utiliser l'instruction **Do...While** afin de réaliser l'addition comme suit :

VB

```
Dim sum As Integer = 0
Do While sum < 100
  sum = sum + 10
Loop
```

Dans le code ci-dessus, la ligne `Do While` évalue la variable `sum` pour déterminer si sa valeur est inférieure à 100 : dans l'affirmative, la ligne de code suivante s'exécute ; dans la négative, le programme passe à la ligne de code qui suit `Loop`. Le mot clé `Loop` demande au code de revenir à la ligne `DoWhile` et d'évaluer la nouvelle valeur de `sum`.

Essayez !

Pour utiliser une instruction Do...While

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `DoWhile`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Depuis la **Boîte à outils**, faites glisser un contrôle **TextBox** et un contrôle **Button** dans le formulaire.
5. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.
6. Dans le gestionnaire d'événements **Button1_Click**, tapez le code suivant :

   VB

   ```
   Dim sum As Integer = 0
   Dim counter As Integer = 0
   Do While sum < 100
     sum = sum + CInt(Textbox1.Text)
     counter = counter + 1
   Loop
   MsgBox("The loop has run " & CStr(counter) & " times!")
   ```
7. Appuyez sur F5 pour exécuter le programme.
8. Dans la zone de texte, tapez un nombre, puis cliquez sur le bouton.

   Un message s'affiche et indique à combien de reprises le nombre a été ajouté à lui-même avant d'atteindre la valeur 100.

9. Dans le menu **Déboguer**, choisissez **Arrêter le débogage** pour terminer le programme. Laissez le projet ouvert. Nous sommes sur le point de lui ajouter des éléments.

Instruction Do...Until

L'instruction **Do...While** répète une boucle tant que la valeur d'une condition reste **True**. Pourtant, vous pouvez parfois souhaiter que votre code se répète jusqu'à ce qu'une condition prenne la valeur **True**. Vous pouvez utiliser l'instruction **Do...Until** comme suit :

VB

```
Dim sum As Integer = 0
Do Until sum >= 100
  sum = sum + 10
Loop
```

Ce code est semblable à celui de l'instruction **Do...While**, si ce n'est que, cette fois, il évalue la variable `sum` pour déterminer si elle est égale ou supérieure à 100.

Essayez !

Pour utiliser une instruction Do...Until

1. Ajoutez le code suivant sous la ligne `MsgBox`.

   VB

   ```
   Dim sum2 As Integer = 0
   Dim counter2 As Integer = 0
   Do Until sum2 >= 100
     sum2 = sum2 + CInt(Textbox1.Text)
     counter2 = counter2 + 1
   Loop
   MsgBox("The loop has run " & CStr(counter2) & " times!")
   ```

2. Appuyez sur F5 pour exécuter le programme.

3. Dans la zone de texte, tapez un nombre, puis cliquez sur le bouton.

   Un deuxième message s'affiche et spécifie à combien de reprises le nombre a été ajouté à lui-même avant d'égaler ou de dépasser la valeur 100.

Étapes suivantes

Dans cette rubrique, vous avez appris comment utiliser les boucles **Do...While** et **Do...Until** pour répéter un code selon des conditions. À ce stade, vous pouvez passer à la leçon suivante, Deux possibilités pour votre programme : instruction If...Then.

Voir aussi

Tâches

Configuration de votre programme pour la répétition d'actions : exécution d'une boucle avec la boucle For...Next

Référence

Do...Loop, instruction (Visual Basic)

Deux possibilités pour votre programme : instruction If...Then

Dans cette leçon, vous apprendrez à utiliser l'instruction **If...Then** pour exécuter un code en fonction de conditions.

Les programmes doivent effectuer des opérations différentes en réaction à des conditions différentes. Par exemple, vous pouvez souhaiter que votre programme vérifie le jour de la semaine, puis effectue une tâche spécifique correspondant au jour identifié. L'instruction **If...Then** vous permet d'évaluer une condition, puis d'exécuter différentes sections d'un code en fonction des résultats de cette condition.

L'exemple suivant illustre le fonctionnement de l'instruction **If...Then**.

VB

```
If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Monday Then
  MsgBox("Today is Monday!")
End If
```

Lorsque ce code s'exécute, la condition (la partie présente entre **If** et **Then**) est évaluée. Si la condition est true, la ligne de code suivante s'exécute et un message s'affiche ; si la condition est false, le code passe à la ligne **End If**. En d'autres termes, le code déclare "Si nous sommes lundi, affiche le message".

Essayez !

Pour utiliser l'instruction If...Then

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `IfThen`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans le gestionnaire d'événements **Form1_Load**, tapez le code suivant.

   VB

   ```
   If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Saturday Or _
     My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Sunday Then
     MsgBox("Happy Weekend!")
   End If
   ```
6. Appuyez sur F5 pour exécuter le programme.

   Si le jour de la semaine est samedi ou dimanche, un message s'affiche et vous souhaite un bon week-end (`Happy Weekend!`). Sinon, aucun message ne s'affiche.
7. Dans le menu **Déboguer**, choisissez **Arrêter le débogage** pour terminer le programme. Laissez le projet ouvert. Vous lui ajouterez des éléments dans la procédure suivante, "Pour utiliser la clause **Else**".

Vous avez pu remarquer dans l'exemple ci-dessus que l'instruction **If...Then** utilisait l'opérateur **Or** pour évaluer plusieurs conditions ("si nous sommes samedi (Saturday) **Or** (ou) si nous sommes dimanche (Sunday)"). Vous pouvez utiliser les opérateurs **Or** et **And** pour évaluer autant de conditions que vous le souhaitez dans une même instruction **If...Then**.

La clause Else

Vous avez appris comment utiliser l'instruction **If...Then** pour exécuter du code si une condition a la valeur true, mais que se passe-t-il si vous souhaitez exécuter un jeu de code alors qu'une condition a la valeur true et une autre a la valeur false ? Dans ce cas, vous pouvez utiliser la clause **Else**. La clause **Else** spécifie un bloc de code qui sera exécuté si la condition est false. L'exemple suivant illustre le fonctionnement de la clause **Else**.

VB

```
If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Friday Then
  MsgBox("Today is Friday!")
Else
  MsgBox("It isn't Friday yet!")
End If
```

Dans cet exemple, l'expression est évaluée ; si elle a la valeur true, la ligne de code suivante est exécutée et le premier message s'affiche. Si elle a la valeur false, le code passe à la clause **Else** et la ligne **Else** suivante est exécutée, entraînant l'affichage du deuxième message.

Essayez !

Pour utiliser la clause Else

1. Changez le code qui figure dans l'instruction **If...Then** en ce qui suit.

   VB

   ```
   If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Saturday Or _
     My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Sunday Then
     MsgBox("Happy Weekend!")
   Else
     MsgBox("Happy Weekday! Don't work too hard!")
   End If
   ```

2. Appuyez sur F5 pour exécuter le programme. Votre programme affiche à présent un message, qui spécifie si nous sommes en semaine ou en week-end, et l'accompagne du contenu approprié.

   Remarque

   Si vous souhaitez tester l'exécution des deux blocs de code, vous pouvez modifier le jour de la semaine en double-cliquant sur l'heure dans la barre des tâches Windows. (La barre des tâches est la barre qui contient le bouton **Démarrer** de Windows ; par défaut, elle est affichée en bas du bureau et l'heure est indiquée dans le coin droit.)

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser l'instruction **If...Then** avec la clause **Else** pour que votre programme exécute sélectivement différents blocs de code en fonction de conditions vérifiées au moment de l'exécution. Pour votre prochaine leçon, vous pouvez choisir de découvrir la manière de sélectionner du code à exécuter en consultant la rubrique Présentation détaillée : utilisation de constructions Select Case pour choisir entre plusieurs options, ou passer à la leçon suivante : Opération à effectuer en cas de problème : gestion des erreurs.

Voir aussi

Tâches

Configuration de votre programme pour la répétition d'actions : exécution d'une boucle avec la boucle For...Next

Comparaisons : utilisation d'expressions pour comparer des valeurs

Référence

If...Then...Else, instruction (Visual Basic)

Présentation détaillée : utilisation de constructions Select Case pour choisir entre plusieurs options

Dans cette leçon, vous apprendrez à utiliser l'instruction **Select Case** pour exécuter un code en fonction de plusieurs conditions.

Dans la leçon précédente, vous avez appris comment utiliser des instructions **If...Then** pour exécuter différents blocs de code selon différentes conditions. Alors qu'il est possible d'évaluer plus de deux conditions dans une même instruction **If...Then** en utilisant le mot clé **ElseIf**, l'instruction **Select Case** constitue un bien meilleur moyen d'évaluer plusieurs conditions.

L'instruction **Select Case** permet d'utiliser autant de conditions (ou cas) que nécessaire, ce qui la rend pratique pour écrire un code destiné à des situations présentant de nombreux choix. Par exemple, supposez que votre programme utilise une variable **String** pour stocker un choix de couleurs et que vous deviez récupérer la valeur chromatique. Le code de l'instruction **Select Case** ressemblerait alors à ce qui suit :

VB

```
Select Case Color
  Case "red"
      MsgBox("You selected red")
  Case "blue"
      MsgBox("You selected blue")
  Case "green"
      MsgBox("You selected green")
End Select
```

Lorsque ce code s'exécute, la ligne **Select Case** détermine la valeur (`Color`) de l'expression. Supposez que `Color` soit une variable **String**, et que cette variable soit un paramètre d'une méthode qui contient l'instruction **Select Case** La valeur de `Color` est alors comparée à la valeur de la première instruction **Case**. Si la valeur correspond, la ligne de code suivante s'exécute, puis le code passe à la ligne **End Select** ; si la valeur ne correspond pas, la ligne **Case** suivante est évaluée.

L'instruction **Case** peut prendre de nombreuses formes (dans l'exemple ci-dessus, il s'agit d'un type **String**). Il peut toutefois s'agir de n'importe quel type de données ou expression.

Vous pouvez évaluer une plage de nombres en utilisant le mot clé **To**, comme suit :

VB

```
Case 1 To 10
```

Dans cet exemple, tout nombre compris entre 1 et 10 aurait pour résultat une correspondance.

Vous pouvez également évaluer plusieurs valeurs dans une instruction **Case** unique en les séparant par des virgules, comme suit :

VB

```
Case "red", "white", "blue"
```

Dans cet exemple, chacune des trois valeurs aurait pour résultat une correspondance.

Vous pouvez également utiliser des opérateurs de comparaison et le mot clé **Is** pour évaluer des valeurs de la manière suivante :

VB

```
Case Is > 9
```

Dans cet exemple, tout nombre supérieur à 9 aurait pour résultat une correspondance.

Case Else

L'exemple ci-dessus fonctionne parfaitement tant que vous connaissez toutes les conditions possibles. Toutefois, que se passe-t-il en présence d'une condition imprévue ? Par exemple, si la valeur de `Color` était `yellow`, le code évaluerait simplement les trois cas, sans rechercher de correspondance, et aucun message ne s'afficherait.

L'instruction **Case Else** permet d'exécuter un code en l'absence de correspondance, comme dans l'exemple suivant :

VB

```
Select Case Color
  Case "red"
      MsgBox("You selected red")
  Case "blue"
      MsgBox("You selected blue")
  Case "green"
      MsgBox("You selected green")
  Case Else
      MsgBox("Please choose red, blue, or green")
End Select
```

Dans le code ci-dessus, si la valeur de `Color` est `yellow`, le programme la compare aux trois premières lignes **Case**, sans rechercher de correspondance. Lorsque la ligne **Case Else** est atteinte, le programme exécute la ligne de code suivante avant de passer à **End Select**.

Pour utiliser l'instruction Select Case

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `SelectCase`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Depuis la **Boîte à outils**, faites glisser un contrôle **TextBox** et un contrôle **Button** dans le formulaire.
5. Double-cliquez sur le bouton pour ouvrir l'éditeur de code.
6. Dans le gestionnaire d'événements **Button1_Click**, tapez le code suivant :

   VB

   ```
   Dim Number As Integer = CInt(Textbox1.Text)
   Select Case Number
     Case 1
         MsgBox("Less than 2")
     Case 2 To 5
         MsgBox("Between 2 and 5")
     Case 6, 7, 8
         MsgBox("Between 6 and 8")
     Case 9 To 10
         MsgBox("Greater than 8")
     Case Else
         MsgBox("Not between 1 and 10")
   End Select
   ```
7. Appuyez sur F5 pour exécuter le programme.
8. Dans la zone de texte, tapez un nombre, puis cliquez sur le bouton.

   Un message s'affiche pour l'instruction **Case** qui correspond au nombre que vous avez saisi.

Étapes suivantes

Dans cette rubrique, vous avez appris comment utiliser l'instruction **Select Case** pour faire un choix parmi plusieurs conditions. À ce stade, vous pouvez passer à la leçon suivante, « Opérations à effectuer en cas de problème : gestion des

erreurs ».

Leçon suivante : Opération à effectuer en cas de problème : gestion des erreurs

Voir aussi

Tâches

Opération à effectuer en cas de problème : gestion des erreurs

Deux possibilités pour votre programme : instruction If...Then

Référence

Select...Case, instruction (Visual Basic)

Opération à effectuer en cas de problème : gestion des erreurs

Dans cette leçon, vous apprendrez comment créer un code de gestion des erreurs de base pour vos programmes.

Même les programmes les mieux conçus rencontrent parfois des erreurs. Certaines erreurs sont des défauts au sein de votre code, qui peuvent être localisés, puis corrigés. D'autres erreurs sont une conséquence normale du fonctionnement du programme ; par exemple, votre programme peut tenter d'ouvrir un fichier qui est déjà en cours d'utilisation. Dans les cas de ce type, les erreurs peuvent être prévues, mais ne peuvent pas être empêchées. En tant que programmeur, votre travail consiste à prévoir ces erreurs et à aider votre programme à les traiter.

Erreurs d'exécution

Une erreur qui se produit pendant qu'un programme est en cours d'exécution est appelée *erreur d'exécution*. Une erreur d'exécution se produit lorsqu'un programme tente d'effectuer une opération pour laquelle il n'a pas été conçu. Par exemple, si votre programme tente d'exécuter une opération illégale, telle que la conversion d'une chaîne non numérique en valeur numérique, une erreur d'exécution se produit.

Lorsqu'une erreur d'exécution se produit, le programme publie une *exception* qui gère les erreurs en recherchant dans le programme le code correspondant. Si aucun code de ce type n'est détecté, le programme s'arrête et doit être redémarré. Cette situation pouvant conduire à une perte de données, il est recommandé de créer un code de gestion des erreurs en tout point où vous prévoyez qu'une erreur est susceptible de se produire.

Bloc Try...Catch...Finally.

Vous pouvez utiliser le bloc **Try...Catch...Finally** pour gérer les erreurs d'exécution au sein de votre code. Vous pouvez appliquer **Try** à un segment de code ; si une exception est générée par ce code, le programme passe au bloc **Catch**. Le code présent dans le bloc **Catch** est alors exécuté. Une fois l'exécution de ce code terminée, tout code présent dans le bloc **Finally** est exécuté. Le bloc **Try...Catch...Finally** entier est fermé par une instruction **End Try** L'exemple suivant illustre l'utilisation de chaque bloc :

VB

```
Try
  ' Code here attempts to do something.
Catch
  ' If an error occurs, code here will be run.
Finally
  ' Code in this block will always be run.
End Try
```

En premier lieu, le code du bloc **Try** est exécuté. S'il s'exécute sans erreur, le programme passe au bloc **Catch** et exécute le code dans le bloc **Finally**. Si une erreur se produit dans le bloc **Try**, l'exécution passe immédiatement au bloc **Catch**. Le code présent dans ce bloc est alors exécuté. Ensuite, le code du bloc **Finally** est exécuté.

Essayez !

Pour utiliser le bloc Try...Catch

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `TryCatch`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Depuis la **Boîte à outils**, faites glisser un contrôle **TextBox** et un contrôle **bouton** dans le formulaire.
5. Double-cliquez sur **Button** pour ouvrir l'éditeur de code.
6. Dans le gestionnaire d'événements **Button1_Click**, tapez le code suivant :

   VB

   ```
   Try
     Dim aNumber As Double = CDbl(Textbox1.Text)
   ```

```
    MsgBox("You entered the number " & aNumber)
  Catch
    MsgBox("Please enter a number.")
  Finally
    MsgBox("Why not try it again?")
  End Try
```

7. Appuyez sur F5 pour exécuter le programme.
8. Dans la zone de texte, tapez une valeur numérique, puis cliquez sur le bouton. Un message affiche le nombre que vous avez saisi suivi d'une invitation à essayer à nouveau.
9. Ensuite, tapez une valeur non numérique dans la zone de texte, telle qu'un mot, puis cliquez sur le bouton. Cette fois, lorsque le programme tente de convertir le texte saisi dans la zone de texte en un nombre, il n'y parvient pas et une erreur se produit. Au lieu de terminer le code du bloc **Try**, le programme exécute le bloc **Catch**, et un message s'affiche et vous invite à saisir un nombre. Le bloc **Finally** s'exécute alors et vous invite à essayer à nouveau.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser des blocs **Try...Catch...Finally** pour créer une structure de gestion des erreurs de base. Cette leçon termine l'introduction au langage Visual Basic. Dans la série de leçons suivante, vous apprendrez comment créer l'interface utilisateur de vos programmes.

Leçon suivante : Création de l'aspect visuel de votre programme : introduction aux Windows Forms

Voir aussi

Tâches

Deux possibilités pour votre programme : instruction If...Then

Connaissance des bogues : trois types d'erreurs de programmation

Référence

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Gestion structurée des exceptions en Visual Basic

Création de l'aspect visuel de votre programme : introduction aux Windows Forms

L'interface utilisateur est la partie de votre programme que les utilisateurs voient lorsqu'ils exécutent le programme. Elle est généralement constituée d'une fenêtre principale ou d'un formulaire, ainsi que de plusieurs contrôles, tels que des boutons ou des champs pour la saisie de texte. Les programmes Visual Basic qui s'exécutent sur votre ordinateur sont connus sous le nom d'applications *Windows Forms*. L'interface utilisateur est créée au moyen de contrôles Windows Forms.

Les leçons de cette section vous montreront comment créer une interface utilisateur à l'aide de certains des contrôles Windows Forms les plus répandus.

Dans cette section

Communication avec l'utilisateur de votre programme : interface utilisateur

Interaction avec l'utilisateur : utilisation des boutons

Affichage et réception de texte : utilisation des étiquettes et des zones de texte

Création d'un programme interactif : création d'un gestionnaire d'événements

Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option

Description : affichage d'images

Proposer des choix à l'utilisateur : création de menus au moment du design

Dans les temps : utilisation de minuteries pour exécuter des actions régulières

Rubriques connexes

Présentation détaillée : partage d'un gestionnaire d'événements

Présentation détaillée : utilisation de plusieurs groupes de cases d'option

Présentation détaillée : davantage d'informations sur les menus

Visite guidée de Visual Basic

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Introduction au langage de programmation Visual Basic

Communication avec l'utilisateur de votre programme : interface utilisateur

Dans cette leçon, vous apprendrez ce qu'est une *interface utilisateur*, ce que sont des contrôles, et comment les ajouter à une interface utilisateur.

Aux débuts de l'informatique personnelle, les utilisateurs interagissaient avec les programmes essentiellement par le biais d'une ligne de commande. L'exécution d'un programme démarrait, s'interrompait pour recevoir l'entrée d'un utilisateur, puis reprenait. En revanche, la plupart des programmes que vous utilisez aujourd'hui s'exécutent dans une ou plusieurs fenêtres qui permettent à l'utilisateur de communiquer, ou de s'*interfacer*, avec le programme par différents biais, notamment au moyen de saisies au clavier, en cliquant sur des boutons ou en choisissant les éléments de menus prédéfinis. Dans cette leçon et celles qui suivent, vous apprendrez comment développer vos propres interfaces utilisateur Windows.

Utilisation de formulaires

Les formulaires sont les blocs de construction de base de votre interface utilisateur. Chaque formulaire de votre programme représente une fenêtre se présente aux utilisateurs. Lorsque vous travaillez dans l'environnement de développement intégré (IDE) de Visual Basic, un formulaire constitue le *concepteur* que vous utilisez pour concevoir votre interface utilisateur ; son utilisation est très semblable à celle de Windows Paint pour dessiner une image.

Les contrôles sont utilisés dans le concepteur pour créer l'aspect de votre interface utilisateur. Un contrôle est un objet dont le comportement et l'aspect sont préalablement définis. Par exemple, un contrôle Button a le même aspect qu'un bouton de commande et se comporte de la même manière : lorsqu'un utilisateur clique dessus, son apparence change pour montrer qu'il a été activé.

En Visual Basic, chaque contrôle a sa propre fonction. Par exemple, un contrôle TextBox permet d'entrer du texte, alors qu'un contrôle PictureBox permet d'afficher des images. Il y a plus de cinquante contrôles différents dans Visual Basic ; vous pouvez également créer vos propres contrôles, connus sous le nom de *contrôles utilisateur*. Vous en saurez plus sur les différents types de contrôle dans les leçons ultérieures.

Lorsque vous concevez votre interface utilisateur, vous faites glisser des contrôles depuis la **Boîte à outils** jusqu'à un formulaire, puis vous les positionnez et les redimensionnez pour obtenir l'aspect souhaité. Vous pouvez modifier plus avant leur aspect en définissant les propriétés des formulaires et des contrôles dans la fenêtre **Propriétés**. Par exemple, les formulaires et la plupart des contrôles disposent d'une propriété **BackColor**. Celle-ci est utilisée pour définir leur couleur d'arrière-plan.

Les propriétés sont également utilisées pour définir le comportement d'un formulaire ou d'un contrôle. Par exemple, la propriété ShowInTaskbar d'un formulaire détermine si le formulaire s'affichera dans la barre des tâches Windows lors de l'exécution du programme. Les propriétés vous permettent de personnaliser l'aspect et le comportement de votre interface utilisateur.

Essayez !

Pour modifier les propriétés d'un formulaire

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `FirstForm`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms est créé. Un nouveau formulaire s'affiche dans la fenêtre principale, et ses propriétés sont visibles dans la fenêtre **Propriétés**, dans le coin inférieur droit de l'IDE de Visual Basic.
4. Cliquez une fois sur le formulaire pour le sélectionner.
5. Dans la fenêtre **Propriétés**, modifiez la propriété Text pour lui affecter la valeur "My First Form", puis appuyez sur ENTRÉE.

   Le texte du haut du formulaire change après la saisie de la nouvelle valeur.
6. Dans la fenêtre **Propriétés**, modifiez la propriété BackColor pour lui affecter une couleur différente en la sélectionnant dans la liste déroulante.

   Remarquez que vous pouvez modifier la propriété **BackColor** à l'aide d'une interface spéciale. Cette interface vous

permet de visualiser la couleur avant de la sélectionner et de choisir parmi différentes couleurs actuellement utilisées par votre système, des couleurs Web standard, ou une sélection de couleurs plus personnalisée. Vous pouvez aussi simplement taper le nom de la couleur (par exemple, `Red`) dans la zone dans la fenêtre **Propriétés**.

Mettez cette procédure en pratique en modifiant d'autres propriétés du formulaire. Lorsque vous êtes prêt, passez à la procédure suivante.

Ajout de contrôles à votre formulaire

Dans cette procédure, vous ajouterez des contrôles à votre formulaire en les sélectionnant dans la fenêtre **Boîte à outils**, habituellement placée sur le côté gauche de l'IDE de Visual Basic, puis en les faisant glisser jusqu'à votre formulaire. Vous manipulerez ensuite les propriétés des contrôles.

Pour ajouter des contrôles à votre formulaire

1. Depuis la **Boîte à outils**, faites glisser dans le formulaire des contrôles **Button**, **TextBox**, Label, puis CheckBox .
2. Sélectionnez le contrôle **Button** et déplacez le dans le formulaire pour modifier son emplacement.

   Remarquez les indications qui s'affichent lorsque vous le faites glisser près des autres contrôles. Ces indications peuvent vous aider à positionner les contrôles avec précision.
3. Répétez le processus avec les autres contrôles jusqu'à ce que l'interface utilisateur ait l'aspect désiré.
4. Sélectionnez le contrôle **Button**. Ensuite, cliquez sur le carré blanc et faites-le glisser dans son coin inférieur droit pour en modifier la taille.
5. Prenez le temps de manipuler les propriétés des contrôles. Cliquez sur chaque contrôle du formulaire pour le sélectionner, puis modifiez certaines de ses propriétés dans la fenêtre **Propriétés**. Les propriétés que vous pouvez essayer de modifier comptent **Font**, **BackColor**, **ForeColor** et **Text**.
6. Appuyez sur F5 pour exécuter le programme. Une fenêtre s'affiche avec les contrôles que vous venez d'ajouter. Vous pouvez cliquer sur le bouton, activer et désactiver la case à cocher, et entrer des informations dans la zone de texte.

Étapes suivantes

Dans cette leçon, vous avez appris comment créer un formulaire et comment lui ajouter des contrôles. Vous avez également appris comment modifier les propriétés de formulaires et de contrôles dans la fenêtre **Propriétés**. Dans les leçons suivantes, nous examinerons plus en détail certains contrôles.

Leçon suivante : Interaction avec l'utilisateur : utilisation des boutons

Voir aussi

Concepts

Présentation détaillée : explication des propriétés, des méthodes et des événements

Autres ressources

Création de l'aspect visuel de votre programme : introduction aux Windows Forms

Interaction avec l'utilisateur : utilisation des boutons

Dans cette leçon, vous apprendrez comment ajouter un contrôle Button à un formulaire, comment modifier l'aspect du bouton correspondant, et comment écrire un code qui s'exécute lorsque ce bouton est activé.

Les boutons constituent le moyen le plus facile pour l'utilisateur d'interagir avec votre programme. Par exemple, nombre de programmes disposent d'un bouton **Quitter**. Comme vous l'avez vu dans la leçon précédente, le contrôle **Button** en Visual Basic se présente et fonctionne comme un bouton de commande. Le contrôle **Button** dispose également d'événements prédéfinis qui peuvent être utilisés pour initialiser des actions, telles que la fermeture d'un programme.

Utilisation des boutons

Les boutons sont généralement des contrôles rectangulaires dont l'aspect présente un relief sur le formulaire. Toutefois, de nombreuses propriétés peuvent être définies pour modifier l'aspect des boutons. La plus évidente est la propriété Text, qui détermine le texte affiché. Ce texte s'affiche dans la *police*, les caractères, déterminée par la propriété Font. La propriété BackColor détermine la couleur du bouton, tandis que la propriété ForeColor détermine la couleur du texte.

Lorsque l'utilisateur clique sur un bouton au moment de l'exécution, le **Button** déclenche l'événement Click. Lorsqu'un événement se produit, les contrôles exécutent un code en réaction à cet événement. Vous pouvez écrire du code qui s'exécute en créant un gestionnaire d'événements.

Un gestionnaire d'événements est une méthode qui s'exécute lorsqu'un événement se produit. Lorsqu'un utilisateur clique sur un bouton, l'événement **Click** du bouton dispose d'un gestionnaire d'événements. Sa programmation est plus facile qu'elle n'en a l'air. Vous apprendrez comment développer un gestionnaire d'événements dans l'exemple ci-dessous. Les événements et les gestionnaires d'événements seront examinés plus en détail dans
Création d'un programme interactif : création d'un gestionnaire d'événements.

Essayez !

Pour utiliser des boutons

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `ButtonExample`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle **Button** dans le formulaire.
5. Dans la fenêtre **Propriétés**, modifiez la propriété **Text** comme suit : `What time is it?`, puis appuyez sur **ENTRÉE**.

   Remarquez que le texte n'est pas adapté à la taille du bouton.
6. Dans la fenêtre **Propriétés**, sélectionnez la propriété AutoSize et affectez-lui la valeur `True`.

   Le bouton se redimensionne pour accueillir le texte.
7. Dans le formulaire, double-cliquez sur le bouton pour ouvrir l'éditeur de code.

   L'éditeur de code s'ouvre au milieu d'une méthode appelée **Button1_Click**. Il s'agit du gestionnaire d'événements **Button1.Click**. Le code que vous écrirez ici s'exécute lorsque le bouton est activé.
8. Dans le gestionnaire d'événements **Button1_Click**, tapez la ligne de code suivante.

   VB

   ```
   MsgBox("The current time is " & Now.ToShortTimeString)
   ```
9. Appuyez sur F5 pour exécuter le programme.

   Le programme s'exécute et le formulaire s'affiche. Lorsque vous cliquez sur le **Button**, un message affiche l'heure actuelle.

Étapes suivantes

Dans cette leçon, vous avez appris comment ajouter un bouton à un formulaire et comment ajouter un code qui s'exécute lorsque l'utilisateur clique sur ce bouton avec la souris. Dans la leçon suivante, vous apprendrez comment travailler avec les contrôles conçus pour afficher et recevoir du texte : Label et TextBox.

Leçon suivante : Affichage et réception de texte : utilisation des étiquettes et des zones de texte

Voir aussi

Tâches

Communication avec l'utilisateur de votre programme : interface utilisateur

Référence

Vue d'ensemble du contrôle Button (Windows Forms)

Concepts

Présentation détaillée : explication des propriétés, des méthodes et des événements

Autres ressources

Création de l'aspect visuel de votre programme : introduction aux Windows Forms

Affichage et réception de texte : utilisation des étiquettes et des zones de texte

Dans cette rubrique, vous apprendrez à utiliser les contrôles Label et TextBox pour afficher du texte et accepter celui d'une saisie d'utilisateur.

L'une des manières les plus faciles de transmettre des informations à l'utilisateur et d'en recevoir de ce dernier consiste à utiliser du texte. Vous pouvez afficher un texte relatif aux fonctionnalités d'un programme, et pouvez recevoir des données en tant que texte de l'utilisateur, puis les utiliser dans votre programme. Visual Basic fournit deux contrôles conçus pour afficher et recevoir du texte. Il s'agit des contrôles **Label** et **TextBox**.

Affichage de texte avec le contrôle Label

Label est le principal contrôle pour afficher du texte. Dans le formulaire, il se présente sous la forme d'un texte entouré d'une zone rectangulaire. La couleur de cette zone est généralement identique à celle du formulaire. Le contrôle se présente simplement sous la forme d'un texte placé directement sur le formulaire.

Sachant qu'un contrôle **Label** est essentiellement destiné à afficher du texte, les propriétés les plus importantes du contrôle **Label** sont celles qui régissent son aspect. La propriété Text contient le texte affiché dans le contrôle **Label**. La propriété Font détermine la police d'affichage du texte dans la propriété **Text**. La propriété ForeColor détermine la couleur du texte proprement dit, et la propriété BackColor la couleur de la zone qui entoure le texte.

Réception de texte avec le contrôle TextBox

Lorsque vous avez besoin à la fois d'afficher et de recevoir un texte, vous utilisez le contrôle **TextBox**, conçu à cet effet. En plus d'être en mesure d'afficher un texte, le contrôle **TextBox** permet, au moment de l'exécution, à l'utilisateur d'entrer un texte dans l'élément **TextBox** et au programme de récupérer ce texte.

Comme dans le cas du contrôle **Label**, les propriétés importantes pour le contrôle **TextBox** sont celles qui sont liées à l'aspect. La propriété **Text** revêt une importance toute particulière. Elle représente le texte placé dans le contrôle **TextBox**. Lorsqu'un utilisateur effectue une saisie dans le contrôle **TextBox**, la propriété **Text** est mise à jour pour refléter les modifications apportées. Ainsi, le texte qui s'affiche dans le contrôle **TextBox** reflète systématiquement la valeur de la propriété **Text**.

Certaines propriétés affectent le comportement du contrôle **TextBox**. La propriété Multiline détermine si le contrôle **TextBox** autorise plusieurs lignes. Si cette propriété a la valeur **False**, la hauteur du contrôle **TextBox** est systématiquement d'une ligne et une seule, et ne peut pas être augmentée. Si la valeur attribuée est **True**, le contrôle **TextBox** autorise plusieurs lignes et sa hauteur est fonction de vos besoins.

Essayez !

Pour créer une interface utilisateur avec les contrôles Label et TextBox

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `TextBoxExample`, puis cliquez sur OK.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle **TextBox**, **Label** et Button jusqu'au formulaire.
5. Sélectionnez le contrôle **Label** et faites-le glisser au-dessus du contrôle **TextBox**
6. Dans la fenêtre **Propriétés**, remplacez la propriété Text du contrôle **Label** par le code suivant.

   ```
   Enter your name and click the button.
   ```

Maintenant que vous avez créé une interface utilisateur de base, vous devez ajouter un peu de code à votre programme. Il sera alors prêt à être testé !

Pour ajouter du code et tester votre programme

1. Double-cliquez sur le contrôle **Button** pour ouvrir l'éditeur de code.

   L'éditeur de code ouvre le gestionnaire d'événements **Button1_Click**.

2. Ajoutez la ligne de code suivante au gestionnaire d'événements **Button1_Click**.

   VB

   ```
   MsgBox("Your Name is " & Textbox1.Text)
   ```

3. Appuyez sur F5 pour exécuter le programme.

4. Lorsque le formulaire apparaît, tapez votre nom dans le contrôle **TextBox** et cliquez sur le bouton. Un message s'affiche et présente le texte dans le contrôle **TextBox**. Modifiez le texte et cliquez à nouveau sur le bouton. Chaque fois vous cliquez sur le bouton, le texte mis à jour s'affiche.

Étapes suivantes

Dans cette rubrique, vous avez appris ce que sont les contrôles **Label** et **TextBox**, et leur utilisation pour afficher et recevoir le texte. Dans la rubrique suivante, vous apprendrez comment créer des méthodes qui gèrent les événements d'un contrôle. Bien que vous ayez déjà appris comment créer des gestionnaires d'événements de base, tels que le gestionnaire d'événements **Button_Click**, vous apprendrez comment créer des méthodes pour gérer certains autres événements qui s'appliquent aux contrôles.

Leçon suivante : Création d'un programme interactif : création d'un gestionnaire d'événements

Voir aussi

Tâches

Interaction avec l'utilisateur : utilisation des boutons

Référence

Vue d'ensemble du contrôle Label (Windows Forms)

Vue d'ensemble du contrôle TextBox (Windows Forms)

Création d'un programme interactif : création d'un gestionnaire d'événements

Dans cette leçon, vous apprendrez comment créer un gestionnaire d'événements.

Comme vous l'avez vu dans les leçons précédentes, les contrôles disposent de propriétés, de méthodes et d'événements, et sont utilisés pour créer l'interface utilisateur. Les événements sont des composants intéressants. Ils surviennent et s'appliquent à un contrôle. Par exemple, un contrôle peut être activé d'un clic de souris, un texte peut y être entré ou le pointeur de la souris peut le survoler.

Lorsqu'un événement pertinent se produit, le contrôle *déclenche un événement*, ce qui signifie qu'il envoie un signal au programme pour lui faire de savoir qu'un événement s'est produit. Le programme vérifie alors qu'il dispose bien de toutes les méthodes nécessaires pour gérer cet événement. Ces méthodes sont appelées *gestionnaires d'événements*. La méthode qui s'exécute lorsqu'un bouton est activé, notamment celle que vous avez créée dans Interaction avec l'utilisateur : utilisation des boutons, en est un exemple.

Vous pouvez créer des gestionnaires d'événements pour différents événements de contrôle. Dans cette leçon, vous créerez des gestionnaires d'événements pour gérer les événements MouseEnter et MouseLeave d'un bouton ; ces événements se produisent lorsque le pointeur d'une souris passe sur un contrôle.

Essayez !

Pour gérer l'événement MouseEnter

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `EventHandler`, puis cliquez sur OK.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle Button dans le formulaire.
5. Dans la fenêtre **Propriétés**, affectez à la propriété AutoSize la valeur **True**.
6. Dans le menu **Affichage**, choisissez **Code** pour ouvrir l'éditeur de code.

   Juste au-dessus de l'éditeur de code, remarquez les deux listes déroulantes. La zone de gauche contient une liste de tous les contrôles présents sur le formulaire, ainsi que **Form1**, (**Général**) et (**Form1 Events**). La zone de droite répertorie chacun des événements disponibles pour l'élément qui figure dans la zone de gauche.
7. Dans la zone de gauche, choisissez **Button1**.
8. Dans la zone de droite, choisissez **MouseEnter**.

   Un nouveau gestionnaire d'événements, appelé **Button1_MouseEnter**, s'affiche dans l'éditeur de code.
9. Dans le gestionnaire d'événements **Button1_MouseEnter**, tapez le code suivant.

   VB

   ```
   Button1.Text = "The Mouse has entered"
   ```
10. Appuyez sur F5 pour exécuter l'application. Passez le pointeur de la souris sur le bouton. Remarquez que lorsque le pointeur de la souris passe sur **Button1**, le texte du bouton change.

Ajout d'un autre gestionnaire d'événements

Vous avez pu remarquer dans l'exemple précédent que, si le texte de **Button1** change lorsque le pointeur de la souris passe sur ce contrôle, le texte d'origine n'est pas rétabli lorsque le pointeur de la souris s'éloigne du contrôle. Si vous souhaitez que le texte change à nouveau lorsque le pointeur de la souris quitte le bouton, vous devez gérer l'événement **MouseLeave** en plus de l'événement **MouseEnter**.

Pour gérer l'événement MouseLeave

1. Dans l'éditeur de code, assurez-vous que **Button1** est bien sélectionné dans la liste déroulante de gauche, puis sélectionnez **MouseLeave** dans celle de droite.

   Un nouveau gestionnaire d'événements, appelé **Button1_MouseLeave**, s'affiche dans l'éditeur de code.

2. Dans le gestionnaire d'événements **Button1_MouseLeave**, tapez le code suivant :

   VB

   ```
   Button1.Text = "The mouse has left"
   ```

3. Appuyez sur F5 pour exécuter l'application.

   Désormais, lorsque le pointeur de la souris passe sur le bouton, le texte devient `The mouse has entered`, mais lorsque le pointeur quitte le bouton, le texte redevient `The mouse has left`.

Étapes suivantes

Dans cette leçon, vous avez appris comment créer un gestionnaire d'événements à l'aide de l'éditeur de code. À ce stade, vous pouvez passer à la leçon suivante de la séquence,
Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option, ou examiner des gestionnaires d'événements avec Présentation détaillée : partage d'un gestionnaire d'événements. Si vous choisissez la deuxième option, vous devez enregistrer votre projet EventHandler pour l'utiliser dans cette leçon.

Voir aussi
Tâches
Affichage et réception de texte : utilisation des étiquettes et des zones de texte
Référence
Vue d'ensemble du contrôle Label (Windows Forms)
Vue d'ensemble du contrôle TextBox (Windows Forms)

Présentation détaillée : partage d'un gestionnaire d'événements

Dans cette leçon, vous apprendrez comment créer un gestionnaire d'événements partagé, qui gère les événements de plusieurs contrôles.

Dans la leçon précédente, Création d'un programme interactif : création d'un gestionnaire d'événements, vous avez appris comment écrire le code qui réagit aux événements MouseEnter et MouseLeave pour un contrôle Button. Que se passe-t-il cependant en présence de plusieurs contrôles **Button** et si vous souhaitez afficher le même message pour chacun d'eux ? Vous pourriez écrire un code dans le gestionnaire d'événements de chaque contrôle. Heureusement, il y a une méthode plus simple.

Si vous examinez attentivement les méthodes du gestionnaire de l'événement **MouseEnter**, vous remarquez que la déclaration **Method** (`Private Sub Button1_MouseEnter(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseEnter`) contient une clause **Handles** (`Handles Button1.MouseEnter`). Il n'est pas étonnant de constater que le mot clé **Handles** spécifie au gestionnaire d'événements les événements qu'il doit gérer.

Pour partager un gestionnaire d'événements entre plusieurs contrôles, vous devez simplement ajouter les noms des contrôles supplémentaires et le nom de l'événement que vous souhaitez gérer. Le gestionnaire d'événements est alors notifié lorsque l'événement se produit pour un de ces contrôles. Par exemple, si vous disposez de deux contrôles **Button**, et si vous souhaitez utiliser le même gestionnaire d'événements pour les deux, la clause **Handles** ressemblera à ce qui suit :

```
Handles Button1.MouseEnter, Button2.MouseEnter.
```

Vous disposez à présent d'une seule méthode qui gère l'événement **MouseEnter** pour les deux contrôles. Mais comment fait le gestionnaire d'événements pour savoir quel contrôle a déclenché cet événement ? Si vous examinez à nouveau la déclaration **Method**, vous remarquez la clause `ByVal sender As Object` : le mot clé **Sender** informe le gestionnaire d'événements de l'identité de l'objet (dans le cas présent, du contrôle) qui a déclenché l'événement.

Essayez !

Pour partager un gestionnaire d'événements

1. Ouvrez le projet EventHandler que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Création d'un programme interactif : création d'un gestionnaire d'événements, et réaliser les procédures qui s'y trouvent.
2. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
3. À partir de la **Boîte à outils**, faites glisser un autre contrôle **Button** dans le formulaire.
4. Dans la fenêtre **Propriétés**, affectez à la propriété AutoSize la valeur **True**.
5. Dans le menu **Affichage**, choisissez **Code** pour ouvrir l'éditeur de code.
6. Dans la déclaration de la méthode **Button1_MouseEnter** (`Private Sub Button1_MouseEnter(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseEnter`), changez la clause **Handles** en `Handles Button1.MouseEnter, Button2.MouseEnter`.
7. Dans le corps de la déclaration d'événement, remplacez le code existant par le code suivant :

   VB

   ```
   If sender.Equals(Button1) Then
     Button1.Text = "The mouse has entered Button1"
   Else
     Button2.Text = "The mouse has entered Button2"
   End If
   ```

   Ce code vérifie que l'expéditeur était bien **Button1**. Dans l'affirmative, la propriété **Text** de **Button1** est mise à jour ; dans la négative, la propriété **Text** de **Button2** est mise à jour.
8. Dans la déclaration de la méthode **Button1_MouseLeave**, changez la clause **Handles** en ce qui suit.

`Handles Button1.MouseLeave, Button2.MouseLeave`.

9. Dans le corps de la déclaration d'événement, remplacez le code existant par le code suivant :

   VB

   ```
   sender.Text = "The mouse has left"
   ```

   Dans ce cas, le code affecte la même chaîne à la propriété **Text** de l'expéditeur (**Button1** ou **Button2**).

10. Appuyez sur F5 pour exécuter l'application.

    Désormais, lorsque le pointeur de la souris passe sur le bouton, le texte devient `The mouse has entered` avec le nom du bouton. Lorsque le pointeur quitte le bouton, le texte redevient `The mouse has left`.

    Essayez d'ajouter d'autres contrôles au formulaire et de modifier les clauses **Handles** pour les inclure ; il n'est même pas nécessaire qu'il s'agisse du même genre de contrôles.

Étapes suivantes

Dans cette leçon, vous avez appris comment partager un seul gestionnaire d'événements entre plusieurs contrôles. Dans la leçon suivante, vous apprendrez comment utiliser deux nouveaux types de contrôles, les contrôles CheckBox et RadioButton, pour présenter des choix aux utilisateurs.

Leçon suivante : Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option

Voir aussi

Tâches

Création d'un programme interactif : création d'un gestionnaire d'événements

Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option

Dans cette leçon, vous apprendrez comment utiliser des cases à cocher et des cases d'option pour présenter et récupérer les choix de l'utilisateur.

Lorsque vous créez l'interface utilisateur de votre programme, vous avez souvent besoin d'un moyen de proposer des choix. Par exemple, supposez vous ayez développé une application pour saisir les commandes d'une pizzeria ; vous pouvez souhaiter permettre aux utilisateurs de sélectionner un ou plusieurs ingrédients. Le contrôle CheckBox constitue une représentation visuelle qui facilite cette possibilité.

Le contrôle **CheckBox** se compose d'une étiquette de texte et d'une case que l'utilisateur peut sélectionner. Lorsque l'utilisateur clique sur la case, une coche s'y affiche. Si la case est à nouveau activée, la coche est supprimée. La propriété **CheckBox.Checked** permet de récupérer l'état de la case à cocher. Si la case présente une coche, la propriété retourne la valeur **True**. Si aucune coche ne s'affiche, la propriété retourne la valeur **False**.

Essayez !

Pour utiliser des cases à cocher

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `UserChoices`, puis cliquez sur OK.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle Button et trois contrôles **CheckBox** dans le formulaire.
5. Dans la fenêtre **Propriétés**, modifiez la propriété Text des cases à cocher **Checkbox1**, **CheckBox2** et **CheckBox3** pour leur attribuer respectivement les valeurs `Pepperoni`, `Sausage` et `Mushrooms`.
6. Dans la fenêtre **Propriétés**, modifiez la propriété **Text** du contrôle **Button1** pour qu'elle affiche `Order Pizza`.
7. Dans le formulaire, double-cliquez sur le bouton pour ouvrir le gestionnaire d'événements **Button1_Click** dans l'éditeur de code.
8. Ajoutez le code suivant au gestionnaire d'événements **Button1_Click** :

   VB

```
Dim toppings As String = ""
If CheckBox1.Checked = True Then
    toppings &= "Cheese "
End If
If CheckBox2.Checked = True Then
    toppings &= "Peppers "
End If
If CheckBox3.Checked = True Then
    toppings &= "Mushrooms"
End If
If toppings <> "" Then
    MsgBox("Your pizza has the following toppings: " & toppings)
End If
```

9. Appuyez sur F5 pour exécuter le programme. Lorsque le formulaire s'affiche, sélectionnez des ingrédients, puis cliquez sur le bouton. Un message qui indique votre choix d'ingrédients s'affiche.

Utilisation de cases d'option pour faire des choix mutuellement exclusifs

Vous venez d'apprendre comment permettre à un utilisateur de choisir une ou plusieurs options. Mais comment faire pour que

l'utilisateur choisisse une seule des options proposées ? Dans ce cas, vous pouvez utiliser le contrôle RadioButton.

Contrairement aux cases à cocher, les cases d'option fonctionnent toujours en groupe. La sélection d'une case d'option désactive instantanément toutes les autres cases d'option du groupe. La présence d'un groupe de cases d'option indique à l'utilisateur qu'il ne peut faire qu'un seul choix parmi ceux qui lui sont proposés.

Vous pouvez utiliser des groupes de contrôles **RadioButton** pour permettre à l'utilisateur de choisir parmi plusieurs options mutuellement exclusives. Par exemple, vous pouvez permettre aux utilisateurs de choisir une sauce normale ou une sauce épicée sur leur pizza, mais pas les deux. Comme dans le cas du contrôle **CheckBox** , vous pouvez obtenir des informations sur l'état du contrôle **RadioButton** à partir de la propriété **RadioButton.Checked**.

Pour utiliser des cases d'option

1. À partir de la **Boîte à outils**, faites glisser deux contrôles **RadioButton** dans le formulaire.
2. Dans la fenêtre **Propriétés**, attribuez à la propriété **Text** de l'élément **RadioButton1** la valeur `Regular Sauce`.
3. Attribuez à la propriété Checked de l'élément **RadioButton1** la valeur **True**.

Conseil
Lorsque vous définissez un groupe de choix, vous devez systématiquement définir un choix comme valeur par défaut.

4. Dans la fenêtre **Propriétés**, attribuez à la propriété **Text** de l'élément **RadioButton2** la valeur `Spicy Sauce`.
5. Dans le formulaire, double-cliquez sur le bouton pour ouvrir le gestionnaire d'événements **Button1_Click** dans l'éditeur de code.
6. Dans le gestionnaire d'événements **Button1_Click**, ajoutez le code suivant :

   VB

   ```
   If RadioButton1.Checked = True Then
       MsgBox("You chose regular sauce")
   Else
       MsgBox("You chose spicy sauce")
   End If
   ```

7. Appuyez sur F5 pour exécuter le programme. Choisissez l'une des cases d'option, puis cliquez sur le bouton **Order Pizza**. Un message indiquant votre choix s'affiche.

   Essayez de sélectionner les deux cases d'option en même temps. Remarquez que les cases d'option sont exclusives. Lorsque vous cliquez sur l'une d'elles, l'autre est automatiquement désactivée.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser les contrôles **CheckBox** et **RadioButton** pour permettre à l'utilisateur de faire des choix dans l'interface utilisateur. À ce stade, vous pouvez passer à la leçon suivante Description : affichage d'images, ou apprendre comment créer plusieurs groupes de cases d'option avec
Présentation détaillée : utilisation de plusieurs groupes de cases d'option. Si vous choisissez la deuxième possibilité, vous devez enregistrer votre projet UserChoices, car vous l'utiliserez dans la leçon suivante.

Voir aussi
Référence
Vue d'ensemble du contrôle CheckBox (Windows Forms)
Vue d'ensemble du contrôle RadioButton (Windows Forms)

Présentation détaillée : utilisation de plusieurs groupes de cases d'option

Dans cette leçon, vous apprendrez comment créer plusieurs groupes de cases d'option mutuellement exclusives sur un même formulaire.

Dans la leçon précédente, vous avez appris comment créer un groupe de cases d'option pour présenter un jeu de choix mutuellement exclusifs à l'utilisateur. Mais que se passe-t-il si vous présentez plusieurs jeux de tels choix ? Vous allez constater que tous les contrôles RadioButton d'un formulaire sont traités comme un seul et même groupe, autorisant ainsi la sélection d'une seule case d'option.

Heureusement, Visual Basic dispose de plusieurs contrôles, connus sous le nom approprié de *contrôles conteneurs*, qui peuvent contenir d'autres contrôles. En plaçant un contrôle conteneur sur le formulaire, puis en plaçant des contrôles **RadioButton** à l'intérieur de ce contrôle conteneur, vous disposez de plusieurs groupes de cases d'option sur le même formulaire.

Les contrôles conteneurs les plus répandus sont GroupBox et Panel. La différence principale entre les deux tient dans ce que le contrôle **GroupBox** dispose d'une bordure visible, tandis que **Panel** en est dépourvu. Lors de l'utilisation d'un contrôle conteneur pour grouper des cases d'option, le contrôle **GroupBox** constitue le meilleur choix, car la bordure fournit une aide visuelle qui indique le regroupement des choix.

Essayez !

Pour utiliser un contrôle GroupBox en tant que conteneur

1. Ouvrez le projet UserChoices que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente,
   Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option, et réaliser les procédures qu'elle contient.
2. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
3. À partir de la **Boîte à outils**, faites glisser un contrôle **GroupBox** dans le formulaire.
4. Dans la fenêtre **Propriétés**, changez la propriété Text du contrôle **GroupBox** en `Select a crust`.
5. Tandis que le contrôle **GroupBox** est encore sélectionné, faites glisser deux contrôles **RadioButton** depuis la **Boîte à outils** et lâchez-les sur le contrôle **GroupBox**.
6. Dans la fenêtre **Propriétés**, changez respectivement les propriétés Text de **RadioButton3** et **RadioButton4** en `Thin crust` et `Thick crust`.
7. Dans le formulaire, double-cliquez sur le bouton **Order Pizza** pour ouvrir le gestionnaire d'événements **Button1_Click** dans l'éditeur de code.
8. Dans le gestionnaire d'événements **Button1_Click**, ajoutez le code suivant.

   VB

   ```
   If RadioButton3.Checked = True Then
       MsgBox("You chose a thin crust")
   Else
       MsgBox("You chose a thick crust")
   End If
   ```

9. Appuyez sur F5 pour exécuter le programme. Choisissez l'une des cases d'option, puis cliquez sur le bouton **Order Pizza**. Un message indiquant votre choix s'affiche. Remarquez que votre choix de sauce est conservé.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser un contrôle conteneur pour grouper des contrôles **RadioButton**. Dans la leçon suivante, vous apprendrez comment afficher des images.

Leçon suivante : Description : affichage d'images

Voir aussi

Tâches

Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option

Comment : regrouper des contrôles au moyen du contrôle GroupBox Windows Forms

Référence

Vue d'ensemble du contrôle Panel (Windows Forms)

Description : affichage d'images

Dans cette leçon, vous apprendrez comment utiliser un contrôle PictureBox pour afficher des images, et également comment afficher une image en tant qu'arrière-plan d'un formulaire.

On dit qu'une image vaut mieux qu'un long discours ; en fait, nombre de programmes utilisent des images pour véhiculer des informations. Il y a plusieurs manières d'afficher des images en Visual Basic ; la plus répandue consiste à utiliser un contrôle **PictureBox**.

Le contrôle **PictureBox** agit en tant que conteneur d'images ; vous choisissez l'image à afficher en définissant la propriété Image. La propriété **Image** peut être définie dans la fenêtre **Propriétés**. Vous pouvez également écrire le code pour signifier au programme l'image à afficher.

Quelques autres propriétés utiles du contrôle **PictureBox** sont la propriété AutoSize, qui détermine si le **PictureBox** s'étire pour s'adapter à la taille de l'image, et la propriété SizeMode, qui s'utilise pour étirer ou centrer l'image, ou encore lui appliquer un zoom, dans le **PictureBox**.

Avant d'ajouter une image à un contrôle **PictureBox**, vous allez généralement ajouter le fichier image à votre projet en tant que *ressource*. Une fois une ressource ajoutée à votre projet, vous pouvez la réutiliser autant de fois que vous le souhaitez ; par exemple, vous pouvez souhaiter afficher la même image à plusieurs endroits.

Essayez !

Pour ajouter une image en tant que ressource

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Pictures`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Dans la fenêtre **Explorateur de solutions**, double-cliquez sur le nœud **Mon projet** pour ouvrir le **Concepteur de projets**.
5. Dans le **Concepteur de projets**, cliquez sur l'onglet **Ressources**.
6. Cliquez sur **Ajouter une ressource**, puis choisissez **Ajouter un fichier existant** dans la liste déroulante.

   La boîte de dialogue **Ajouter le fichier existant aux ressources** s'affiche. Si aucun fichier image ne s'affiche, naviguez jusqu'à un dossier qui contient des images.
7. Sélectionnez un fichier image (avec l'extension .bmp, .gif ou .jpg), puis cliquez sur **Ouvrir**. Dans cet exemple, il est recommandé de choisir une petite image.

   L'image est ajoutée à votre projet et s'affiche dans la fenêtre **Gestionnaire des ressources**.
8. Répétez les deux étapes précédentes pour ajouter une deuxième image à votre projet.
9. Dans le menu **Fichier**, choisissez **Fermer**. Si vous êtes invité à enregistrer des modifications, choisissez **Oui**.

Pour afficher des images à l'aide d'un contrôle PictureBox

1. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
2. À partir de la **Boîte à outils**, faites glisser un contrôle **PictureBox** dans le formulaire.
3. Dans la fenêtre **Propriétés**, cliquez sur le bouton Sélection (**...**) placé en regard de la propriété **Image** pour ouvrir la boîte de dialogue **Sélectionner une ressource**.
4. Dans la liste **Entrée**, choisissez l'une des images que vous avez ajoutées, puis cliquez sur **OK**.
5. Sélectionnez la propriété **SizeMode** et affectez-lui la valeur **AutoSize**.

   Remarquez la manière dont le contrôle **PictureBox** se redimensionne automatiquement pour s'adapter à l'image.
6. Dans le formulaire, double-cliquez sur le contrôle **PictureBox** pour ouvrir le gestionnaire d'événements

PictureBox1_Click dans l'éditeur de code.

7. Ajoutez le code suivant au gestionnaire d'événements **PictureBox1_Click** :

Remarque
Vous devez remplacer "`MyPictureName2`" par le nom réel de la deuxième image ajoutée auparavant.

   VB

   ```
   PictureBox1.Image = My.Resources.MyPictureName2
   ```

8. Appuyez sur F5 pour exécuter le programme. Lorsque le formulaire s'affiche, cliquez sur l'image pour en faire la deuxième image à s'afficher.

Affichage d'une image d'arrière-plan sur un formulaire

En plus d'afficher une image dans un contrôle **PictureBox**, vous pouvez également afficher une image en tant qu'arrière-plan de votre formulaire. La propriété BackgroundImage d'un formulaire permet d'afficher une image qui apparaît derrière tous autres contrôles sur le formulaire, à la manière d'un papier peint sur le bureau de Windows.

Tout comme Windows vous permet de choisir si le papier peint est centré, en mosaïque, ou étiré pour occuper l'écran, la propriété BackgroundImageLayout permet de faire de même sur un formulaire.

Conseil
Nombre d'autres contrôles, tels que Panel, GroupBox et même Button, disposent également d'une propriété **BackgroundImage**. Essayez-les pour voir !

Essayez !

Pour afficher une image d'arrière-plan sur un formulaire

1. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
2. Sélectionnez le formulaire en cliquant hors de **PictureBox**.
3. Dans la fenêtre **Propriétés**, cliquez sur le bouton Sélection (**...**) placé en regard de la propriété **BackgroundImage** pour ouvrir la boîte de dialogue **Sélectionner une ressource**.
4. Dans la liste **Entrée**, choisissez l'une des images que vous avez ajoutées auparavant, puis cliquez sur **OK**.

   Remarquez que l'image s'affiche sur le formulaire derrière le **PictureBox** et que, par défaut, elle est disposée en mosaïque.

Remarque
Si l'image présente dans le contrôle **PictureBox** est trop grande, vous pouvez ne pas être en mesure de voir l'image d'arrière-plan. Dans ce cas, sélectionnez le contrôle **PictureBox** et faites-le glisser jusqu'au bas du formulaire pour l'écarter.

5. Sélectionnez la propriété **BackgroundImageLayout** et affectez-lui la valeur Stretch.

   Remarquez la manière dont l'image s'étire pour occuper l'intégralité du formulaire.

6. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
7. Assurez-vous que **Form1 Events** est bien sélectionné dans la liste déroulante de gauche, puis sélectionnez **Click** dans celle de droite.
8. Ajouter le code suivant au gestionnaire d'événements **Form1_Click**

   VB

   ```
   If Me.BackgroundImageLayout = ImageLayout.Stretch Then
     Me.BackgroundImageLayout = ImageLayout.Center
   ```

```
  Else
    Me.BackgroundImageLayout = ImageLayout.Stretch
  End If
```

9. Appuyez sur F5 pour exécuter le programme. Lorsque le formulaire s'affiche, cliquez dedans pour modifier la disposition.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser le contrôle **PictureBox** pour afficher des images, et comment utiliser la propriété **BackgroundImage** d'un formulaire. Dans la leçon suivante, vous apprendrez comment créer des menus pour présenter des choix aux utilisateurs.

Leçon suivante : Proposer des choix à l'utilisateur : création de menus au moment du design

Voir aussi

Tâches

Obtention des choix de l'utilisateur : utilisation des cases à cocher et des cases d'option

Comment : définir l'arrière-plan d'un panneau Windows Forms

Référence

Vue d'ensemble du contrôle PictureBox (Windows Forms)

Proposer des choix à l'utilisateur : création de menus au moment du design

Dans cette leçon, vous apprendrez comment créer des menus et comment écrire un code qui s'exécute lorsque des éléments de menu sont sélectionnés.

Les menus constituent pour l'utilisateur un moyen facile et familier de faire des choix quant à votre programme. Les menus sont généralement utilisés pour présenter les options d'un programme, pour ajouter des raccourcis pour les tâches les plus répandues, notamment couper et coller, ou pour charger et enregistrer des fichiers.

Visual Basic facilite l'implémentation des menus. Le contrôle MenuStrip vous permet de créer des menus graphiquement. Une fois placé dans un formulaire, le contrôle **MenuStrip** se présente sous la forme d'une zone. Les mots "Tapez ici" figurent dans la partie supérieure du formulaire. Vous pouvez cliquer dans la zone et y entrer un texte pour créer le titre du menu correspondant.

Lorsque le titre d'un élément de menu est défini, des éléments de menu supplémentaires peuvent être créés sous le premier élément, à sa droite, et fournir autant d'éléments ou de sous-éléments supplémentaires que vous le souhaitez. Lorsque l'aspect de votre menu vous convient, créez les gestionnaires d'événements qui régiront les événements Click associés à chaque élément.

Essayez !

Pour ajouter un menu

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Menus`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle **MenuStrip** dans le formulaire.

   Quel que soit l'emplacement où vous le placez, le contrôle **MenuStrip** se fixe automatiquement au sommet du formulaire.

   Vous avez pu également remarquer qu'une icône **MenuStrip1** est ajoutée dans une zone grise sous le formulaire. Cette zone est appelée barre d'état des composants. Si vous cliquez hors du contrôle **MenuStrip**, la zone disparaît. Vous la rappelez en cliquant sur l'icône **MenuStrip1**.
5. Dans le formulaire, cliquez sur le contrôle **MenuStrip**, tapez `File`, puis appuyez sur **Entrée**.

   De nouvelles zones destinées aux entrées de menu supplémentaires s'affichent sous le premier élément de menu, à sa droite. Il s'agit d'espaces destinés à des éléments de menu supplémentaires. Vous pouvez continuer d'ajouter des éléments de menu dans l'un des deux sens, jusqu'à ce que votre menu soit complet.
6. Dans la zone située sous la première zone, tapez `Exit`, puis appuyez sur **Entrée**.
7. Double-cliquez sur le menu **Quitter** pour ouvrir l'éditeur de code.
8. Dans le gestionnaire d'événements **ExitToolStripMenuItem_Click**, tapez le code suivant.

   VB

   ```
   Application.Exit()
   ```
9. Appuyez sur F5 pour exécuter le programme. À l'aide de la souris, cliquez sur le menu **Fichier**, puis choisissez **Quitter**. Votre application se ferme.

Dans cette leçon, vous avez appris comment utiliser le contrôle **MenuStrip** pour concevoir des menus. À ce stade, vous pouvez poursuivre par la leçon suivante, consacrée aux minuteries, ou examiner des utilisations plus avancées des menus dans Présentation détaillée : davantage d'informations sur les menus, puis poursuivre par la leçon consacrée aux minuteries.

Leçon suivante : Dans les temps : utilisation de minuteries pour exécuter des actions régulières

Voir aussi

Tâches

Description : affichage d'images

Présentation détaillée : davantage d'informations sur les menus

Dans cette leçon, vous apprendrez comment pour activer ou désactiver des menus au moment de l'exécution, et également comment créer des menus contextuels.

Dans la leçon précédente, vous avez appris comment utiliser le contrôle MenuStrip pour créer des menus qui permettent aux utilisateurs de faire des choix relatifs à votre programme. Dans quelques cas, toutefois, les choix peuvent être uniquement disponibles à certains moments. Par exemple, la commande de menu **Copier** est disponible uniquement lorsqu'il y a quelque chose à copier.

La plupart des programmes désactivent les commandes de menu lorsqu'elles sont indisponibles, au lieu de les masquer. Lorsqu'un élément de menu est désactivé, le texte de menu devient gris et cliquer sur l'élément de menu ne produit aucun effet. Lors de l'utilisation d'un contrôle **MenuStrip**, vous pouvez désactiver et activer des éléments de menu à l'aide de la propriété Enabled du MenuItem.

Essayez !

Pour désactiver ou activer des éléments de menu

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Menus2`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle **MenuStrip** et un contrôle TextBox dans le formulaire.
5. Dans le formulaire, cliquez sur le contrôle **MenuStrip** et tapez `Edit,` puis appuyez sur ENTRÉE.
6. Dans la zone située sous la première zone, tapez `Copy`, puis appuyez sur ENTRÉE.
7. Dans la fenêtre **Propriétés**, attribuez à la propriété **Enabled** de **CopyToolStripMenuItem** la valeur **False**.
8. Double-cliquez sur le contrôle **TextBox** pour ouvrir l'éditeur de code.
9. Dans le gestionnaire d'événements **TextBox1_TextChanged**, tapez le code suivant.

   VB

```
If Textbox1.Text <> "" Then
  CopyToolStripMenuItem.Enabled = True
Else
  CopyToolStripMenuItem.Enabled = False
End If
```

10. Appuyez sur F5 pour exécuter le programme. Cliquez sur le menu **Edition**. L'élément de menu **Copier** est désactivé. Effectuez une saisie dans le contrôle **TextBox**, puis cliquez à nouveau sur le menu **Edition**. L'élément de menu **Copier** est à présent activé.

Création de menus contextuels

Beaucoup de programmes utilisent des *menus contextuels* pour faciliter l'accès aux commandes les plus utilisées. Pour accéder à un menu contextuel, cliquez avec le bouton droit sur un formulaire ou un contrôle au moment de l'exécution. Vous pouvez créer vos propres menus contextuels dans Visual Basic, à l'aide d'un contrôle ContextMenuStrip.

Comme avec le contrôle **MenuStrip**, lorsque vous faites glisser un contrôle **ContextMenuStrip** dans un formulaire, le contrôle **ContextMenuStrip** s'affiche dans la partie supérieure du formulaire sous la forme d'une zone qui présente le texte « Tapez ici », et une icône est ajoutée à la barre d'état des composants. Contrairement à **MenuStrip**, les éléments supplémentaires s'ajoutent uniquement sous le premier élément de menu, créant ainsi un menu vertical.

De plus, un **ContextMenuStrip** doit être associé au formulaire ou au contrôle dans lequel vous souhaitez qu'il s'affiche. Pour ce faire, attribuez pour valeur à la propriété **ContextMenuStrip** du formulaire ou du contrôle le nom du contrôle **ContextMenuStrip**. Vous pouvez associer un seul **ContextMenuStrip** avec autant de contrôles que vous le souhaitez.

Essayez !

Pour créer un menu contextuel

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `ContextMenus`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle **ContextMenuStrip** dans le formulaire.
5. Dans la fenêtre **Propriétés**, sélectionnez la propriété **ContextMenuStrip** du formulaire, puis choisissez **ContextMenuStrip1** dans la liste déroulante.
6. Dans le formulaire, cliquez sur le contrôle **ContextMenuStrip** et tapez `Option1`, puis appuyez sur ENTRÉE.
7. Dans la zone située sous la première zone, tapez `Option2`, puis appuyez sur ENTRÉE.
8. Double-cliquez sur l'élément de menu **Option1** pour ouvrir l'éditeur de code.
9. Dans le gestionnaire d'événements **Option1ToolStripMenuItem_Click**, tapez le code suivant.

   VB

   ```
   MsgBox("You chose Option 1")
   ```
10. Dans l'**éditeur de code**, sélectionnez **Option2ToolStripMenuItem** dans la liste déroulante de gauche, puis **Click** dans celle de droite.

    Un nouveau gestionnaire d'événements appelé **Option2ToolStripMenuItem _Click** s'affiche dans l'éditeur de code.
11. Dans le gestionnaire d'événements **Option2ToolStripMenuItem _Click**, tapez le code suivant.

    VB

    ```
    MsgBox("You chose Option 2")
    ```
12. Appuyez sur F5 pour exécuter le programme. Cliquez avec le bouton droit sur le formulaire, puis cliquez sur l'un des éléments du menu contextuel ; un message s'affiche et vous informe de l'option que vous avez choisie.

Étapes suivantes

Dans cette leçon, vous avez appris comment activer et désactiver des menus, et également comment créer un menu contextuel. Dans la rubrique suivante, vous apprendrez comment utiliser un type de contrôle différent, Timer, pour effectuer des actions.

Leçon suivante : Dans les temps : utilisation de minuteries pour exécuter des actions régulières

Voir aussi

Tâches

Proposer des choix à l'utilisateur : création de menus au moment du design

Comment : associer un ContextMenuStrip à un contrôle

Référence

Vue d'ensemble du contrôle MenuStrip (Windows Forms)

Dans les temps : utilisation de minuteries pour exécuter des actions régulières

Dans cette leçon, vous apprendrez comment utiliser le composant Timer pour effectuer des actions non sollicitées par une entrée d'utilisateur.

Vous pouvez parfois trouver utile d'exécuter à plusieurs reprises des actions dans vos programmes (par exemple, enregistrer un fichier à plusieurs minutes d'intervalle ou mettre à jour l'interface utilisateur). Le composant **Timer** vous permet d'exécuter des actions définies de manière régulière sans impliquer aucune entrée de l'utilisateur.

Le composant **Timer** est différent des contrôles que vous avez utilisés jusqu'à présent en ce qu'il ne dispose d'aucune représentation visuelle au moment de l'exécution. Les contrôles dépourvus de représentation visuelle sont connus sous le nom de *composants*. Sachant que l'utilisateur n'a aucun moyen d'interagir directement avec le composant **Timer**, ce dernier s'exécute en arrière-plan tandis que le programme fonctionne.

Deux propriétés et un événement du composant **Timer** sont très souvent utilisés. La propriété Enabled détermine si le composant **Timer** fonctionne. Si **Enabled** a la valeur **True**, **Timer** est actif. Si **Enabled** a la valeur **False**, **Timer** n'est pas actif.

La propriété Interval détermine le nombre de millisecondes qui s'écoulent entre les impulsions du **Timer**. Par exemple, si la propriété **Interval** a pour valeur `1000`, le composant **Timer** déclenche l'événement Tick toutes les 1 000 millisecondes, soit chaque seconde.

L'événement **Tick** est déclenché par le composant **Timer** à intervalles réguliers, selon la valeur de la propriété **Interval**. Vous pouvez ajouter un code à un gestionnaire d'événements **Timer.Tick**. Ce code s'exécute alors lorsque l'événement **Tick** se déclenche.

En définissant les propriétés **Enabled** et **Interval**, et en ajoutant un code au gestionnaire d'événements **Tick**, vous faites en sorte que ce code s'exécute à intervalles réguliers, sans nécessiter aucune intervention de l'utilisateur.

Essayez !

Pour utiliser un composant Timer

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Timer`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser un contrôle Label et un composant **Timer** dans le formulaire.

   Le composant **Timer** ne figure pas sur le formulaire proprement dit, mais se trouve dans la barre d'état des composants, sous le formulaire. En effet, la minuterie ne dispose d'aucune représentation visuelle.
5. Sélectionnez le composant **Timer**, puis dans la fenêtre **Propriétés**, affectez à la propriété **Enabled** la valeur **True** et à la propriété **Interval** la valeur `1000`.
6. Double-cliquez sur le composant **Timer** pour ouvrir l'éditeur de code.
7. Dans le gestionnaire d'événements **Timer1_Tick**, tapez le code suivant.

   VB

   ```
   Label1.Text = My.Computer.Clock.LocalTime.ToLongTimeString
   ```
8. Appuyez sur F5 pour exécuter l'application.

   Le texte de l'étiquette est mis à jour à chaque seconde avec l'heure corrigée.

Étapes suivantes

Dans cette leçon, vous avez appris à utiliser le composant **Timer** pour exécuter un code à intervalles réguliers. Un **Timer** vous permet de programmer un code pour qu'il s'exécute à des intervalles réguliers que vous choisissez. Cette leçon conclut cette

section de la visite guidée.

Dans la section suivante, vous apprendrez comment utiliser les outils de Visual Basic pour localiser et corriger les erreurs (appelées généralement *bogues*) présentes dans votre programme.

Leçon suivante : Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Voir aussi

Tâches

Proposer des choix à l'utilisateur : création de menus au moment du design

Référence

Vue d'ensemble du composant Timer (Windows Forms)

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Lors de l'écriture d'un logiciel, des erreurs peuvent et vont survenir. En cas d'erreur typographique, votre programme peut ne pas se comporter conformément à vos attentes, ou ne pas s'exécuter du tout. En cas d'erreur dans votre programme, vous devez la rechercher et la corriger. Le processus de recherche et de correction des erreurs est appelé *débogage*.

Dans les leçons suivantes, vous apprendrez plusieurs techniques pour déboguer un programme Visual Basic.

Dans cette section

Recherche des erreurs : introduction au débogage Visual Basic

Connaissance des bogues : trois types d'erreurs de programmation

Oups, une faute de frappe : recherche et élimination des erreurs du compilateur

Aïe ! Mon programme n'a pas apprécié ! Recherche et suppression des erreurs d'exécution

Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques

Prise de notes dans vos programmes : utilisation des commentaires

Rubriques connexes

Présentation détaillée : que se passe-t-il si... je teste du code dans la fenêtre Exécution

Crédit supplémentaire : quelque chose ne fonctionne toujours pas

Création de l'aspect visuel de votre programme : introduction aux Windows Forms

Gestion de vos enregistrements : utilisation de données dans votre programme

Visite guidée de Visual Basic

Recherche des erreurs : introduction au débogage Visual Basic

Dans cette leçon, vous apprendrez à corriger les erreurs d'un programme par le biais du débogage.

Peu importe le soin avec lequel vous concevez un programme ou écrivez un code, la présence d'erreurs est inévitable. Il peut arriver que des erreurs empêchent le programme de démarrer, entraînent son arrêt ou son *blocage*, ou le laissent s'exécuter, mais sans fournir les résultats attendus.

Bien sûr, lorsque des erreurs se produisent, vous devez les localiser et les corriger. Les erreurs présentes dans un programme sont communément appelées *bogues*. Le processus qui consiste à les localiser et à les corriger est connu sous le nom de *débogage*.

Le processus du débogage est *itératif* ; vous le répétez à de nombreuses reprises. En général, vous rédigez un code, exécutez le programme jusqu'à ce qu'une erreur se produise, localisez et corrigez le bogue, puis exécutez à nouveau le programme.

Dans la plupart des cas, vous n'avez pas besoin d'arrêter le programme pour le corriger. Vous pouvez corriger le code à l'endroit où l'erreur s'est produite et reprendre l'exécution du programme à l'endroit où vous l'aviez arrêtée ; ce processus se nomme *Modifier & Continuer*.

Le débogage s'effectue dans l'environnement de développement intégré (IDE) Visual Basic, lequel contient plusieurs commandes et fenêtres spéciales pour vous aider à localiser des bogues. Vous en saurez davantage sur ces éléments dans les leçons suivantes.

Essayez !

Remarque

Cet exemple implique une exception. Les exceptions sont des objets qui sont créés (et levés) au moment où le programme se rend compte qu'une erreur s'est produite. Différents types d'exceptions sont créés, selon le type d'erreur qui s'est produite. A vec les paramètres utilisateurs par défaut, si une exception se produit lors de l'exécution de votre programme Visual Basic, u ne boîte de dialogue s'affiche pour expliquer l'erreur et vous aider à la résoudre.

Pour utiliser la fonctionnalité Modifier & Continuer

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Edit`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans le gestionnaire d'événements **Form_Load**, ajoutez le code suivant.

   VB

   ```
   Dim number As Integer = 1
   Dim numbers As String = ""
   MsgBox(numbers + 1)
   ```
6. Appuyez sur F5 pour exécuter le programme. Le programme s'arrête et une boîte de dialogue d'exception s'affiche avec le message « L'exception InvalidCastException n'a pas été gérée ».

   L'exception s'est produite du fait d'une erreur typographique dans le code. La variable incorrecte a été utilisée ; il faut utiliser `number`, la variable **Integer**, et non `numbers`, la variable **String**.

   Remarquez que le programme est toujours en cours d'exécution ; il est seulement passé en mode *arrêt* de débogage. À l'aide de Modifier & Continuer, vous pouvez résoudre l'erreur sans être obligé d'arrêter le programme (et de l'exécuter à nouveau à des fins de vérification).
7. Dans l'éditeur de code, remplacez `numbers + 1` par `number + 1`.

8. Appuyez sur F5 pour continuer. Un message doit s'afficher et présenter le nombre 2.

Étapes suivantes

Dans cette leçon, vous avez appris ce qu'est le débogage, ainsi que la manière de corriger un bogue et de poursuivre l'exécution de votre programme. Dans la leçon suivante, vous apprendrez les différents types d'erreurs.

Leçon suivante : Connaissance des bogues : trois types d'erreurs de programmation

Voir aussi

Autres ressources

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage
Création de l'aspect visuel de votre programme : introduction aux Windows Forms
Visite guidée de Visual Basic

Connaissance des bogues : trois types d'erreurs de programmation

Dans cette leçon, vous apprendrez les différents types d'erreurs qui peuvent se produire lors de l'écriture d'un programme.

Même les programmeurs les plus expérimentés font des erreurs. Savoir comment déboguer une application et rechercher ces erreurs est un thème important en programmation. Cependant, avant d'aborder le processus de débogage, il est utile de connaître les différents types de bogues que vous devrez rechercher et corriger.

Les erreurs de programmation se répartissent dans trois catégories : les *erreurs de compilation*, les *erreurs d'exécution* et les *erreurs logiques*. Les techniques de débogage propres à chaque catégorie seront examinées dans les trois prochaines leçons.

Erreurs de compilation

Les erreurs de compilation, également connues sous le nom d'*erreurs du compilateur*, sont des erreurs qui empêchent votre programme de s'exécuter. Lorsque vous appuyez sur F5 pour exécuter un programme, Visual Basic compile votre code dans un langage binaire que l'ordinateur comprend. Si le compilateur Visual Basic rencontre un code qu'il ne comprend pas, il génère une erreur du compilateur.

La plupart des erreurs du compilateur sont engendrées par celles que vous faites lors de la saisie du code. Par exemple, vous pouvez mal orthographier un mot clé, oublier certains éléments de ponctuation nécessaires ou tenter d'utiliser une instruction **End If** avant une instruction **If**.

Heureusement l'éditeur de code de Visual Basic a été conçu pour identifier ces erreurs avant toute tentative d'exécution du programme. Vous apprendrez comment localiser et résoudre des erreurs de compilation dans la leçon suivante, Oups, une faute de frappe : recherche et élimination des erreurs du compilateur.

Erreurs d'exécution

Les erreurs d'exécution se produisent pendant que votre programme s'exécute. Elles se produisent généralement lorsque votre programme tente une opération impossible.

Par exemple, une division par zéro. Supposez l'instruction suivante :

```
Speed = Miles / Hours
```

Si la variable `Hours` a la valeur 0, l'opération de division échoue et provoque une erreur d'exécution. Le programme doit s'exécuter pour que cette erreur soit détectée. Par ailleurs, si `Hours` contient une valeur valide, aucune erreur ne se produit.

Lorsqu'une erreur d'exécution se produit, les outils de débogage de Visual Basic vous permettent d'en déterminer la cause. Vous apprendrez comment localiser et corriger des erreurs d'exécution dans la leçon Aïe ! Mon programme n'a pas apprécié ! Recherche et suppression des erreurs d'exécution.

Erreurs logiques

Les erreurs logiques empêchent votre programme de faire ce que vous avez prévu qu'il fasse. Votre code peut se compiler et s'exécuter sans erreur, mais une opération peut produire un résultat inattendu.

Par exemple, votre programme peut contenir une variable nommée `FirstName`, dont la valeur initiale est une chaîne vide. Par la suite, votre programme peut concaténer `FirstName` avec une autre variable nommée `LastName` pour afficher un nom complet. Si vous avez oublié d'affecter une valeur à la variable `FirstName`, seul le nom de famille s'affiche, au lieu du nom complet prévu.

Les erreurs logiques sont les plus difficiles à localiser et à corriger. Toutefois, Visual Basic dispose également d'outils de débogage qui facilitent cette tâche. Vous apprendrez comment pour localiser et corriger des erreurs logiques dans Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques.

Étapes suivantes

Dans cette leçon, vous avez appris les trois catégories d'erreurs de programmation. Dans la leçon suivante, vous apprendrez comment déboguer les erreurs du compilateur.

Leçon suivante : Oups, une faute de frappe : recherche et élimination des erreurs du compilateur

Voir aussi

Tâches

Recherche des erreurs : introduction au débogage Visual Basic

Autres ressources

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Oups, une faute de frappe : recherche et élimination des erreurs du compilateur

Dans cette leçon, vous apprendrez comment localiser et corriger des erreurs du compilateur.

Comme vous l'avez appris dans la leçon précédente, les erreurs du compilateur se produisent lorsque le compilateur Visual Basic rencontre un code non reconnaissable ; généralement parce que vous avez fait une erreur de saisie. Etant donné que les erreurs du compilateur empêchent un programme de s'exécuter, vous devez localiser et corriger, ou "déboguer", ces erreurs avant de tenter d'exécuter votre programme.

Localisation et correction des erreurs du compilateur

La recherche d'erreurs du compilateur est en fait assez facile, étant donné que le programme ne peut pas s'exécuter tant que les erreurs ne sont pas corrigées. Lorsque vous appuyez sur F5, en présence d'erreurs du compilateur, une boîte de dialogue s'ouvre et affiche le message "**Des erreurs se sont produites lors de la génération. Voulez-vous continuer ?**". Si vous sélectionnez **Oui**, la dernière version exempte d'erreur du programme s'exécute ; si vous sélectionnez **Non**, le programme s'arrête et la fenêtre **Liste d'erreurs** s'affiche.

La fenêtre **Liste d'erreurs** affiche toutes les informations relatives à l'erreur du compilateur, y compris sa description et son emplacement dans votre code. Si vous double-cliquez sur l'erreur dans la fenêtre **Liste d'erreurs**, la ligne de code incriminée est mise en surbrillance dans l'éditeur de code. Vous pouvez également appuyer sur F1 pour afficher l'Aide et obtenir davantage d'informations sur l'erreur et la manière de la corriger.

L'éditeur de code Visual Basic peut également vous aider à localiser et corriger des erreurs du compilateur avant même que vous ne tentiez d'exécuter le programme. À l'aide d'une fonctionnalité appelée *IntelliSense*, Visual Basic inspecte votre code à mesure que vous l'entrez. S'il détecte un élément de code susceptible d'engendrer une erreur du compilateur, il le souligne d'une ligne bleue ondulée. Si vous maintenez le pointeur de la souris sur cette ligne, un message s'affiche et décrit l'erreur. Si la fenêtre **Liste d'erreurs** est affichée, le message d'erreur s'y affiche également.

Essayez !

Pour localiser et corriger des erreurs du compilateur

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `CompilerErrors`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans le gestionnaire d'événements **Form_Load**, ajoutez le code suivant.

   VB

   ```
   End If
   ```
6. Appuyez sur ENTRÉE. Une ligne bleue ondulée s'affiche sous `End If`.

   Si vous maintenez le pointeur de la souris sur la ligne, le message suivant s'affiche : "'End If' doit être précédé d'un 'If' correspondant."
7. Modifiez le code pour lui donner l'aspect suivant :

   VB

   ```
   If 1 < 2 Then
   End If
   ```

   Notez que la ligne bleue ondulée a disparu.

8. Ajoutez la nouvelle ligne de code suivante après l'instruction If...Then.

   VB

   ```
   MgBox("Hello")
   ```

9. Appuyez sur F5 pour exécuter le programme. Une boîte de dialogue s'ouvre et affiche le message suivant : "**Des erreurs se sont produites lors de la génération. Voulez-vous continuer et exécuter la dernière génération réussie** ?"
10. Cliquez sur **Non**. La fenêtre **Liste d'erreurs** s'ouvre et affiche le message d'erreur suivant : "Le nom 'MgBox' n'est pas déclaré".
11. Double-cliquez sur le message d'erreur dans la **Liste d'erreurs** et remplacez le code par `MsgBox("Hello")`.
12. Appuyez à nouveau sur F5. Cette fois, le programme doit normalement s'exécuter, et un message s'affiche.

Étapes suivantes

Dans cette leçon, vous avez appris comment localiser et corriger des erreurs du compilateur. Dans la leçon suivante, vous apprendrez comment corriger un type d'erreur différent ; les erreurs d'exécution. Leçon suivante :
Aïe ! Mon programme n'a pas apprécié ! Recherche et suppression des erreurs d'exécution

Voir aussi
Tâches
Connaissance des bogues : trois types d'erreurs de programmation
Recherche des erreurs : introduction au débogage Visual Basic

Aïe ! Mon programme n'a pas apprécié ! Recherche et suppression des erreurs d'exécution

Dans cette leçon, vous allez apprendre à déboguer votre programme et à corriger des erreurs d'exécution.

Comme vous l'avez appris auparavant, les erreurs d'exécution se produisent lorsque votre programme tente d'exécuter une opération impossible. Lorsqu'une erreur d'exécution se produit, le programme s'arrête et un message d'erreur s'affiche ; pour que le programme puisse se poursuivre, vous devez déboguer l'erreur et la corriger.

Recherche et correction des erreurs d'exécution

La plupart des erreurs d'exécution se produisent du fait de la présence d'une erreur dans votre code ; par exemple, lorsque vous oubliez d'assigner une valeur à une variable avant de l'utiliser. Lorsque vous exécutez votre programme, si une erreur est identifiée, ce dernier s'arrête et la boîte de dialogue **Assistant Exception** s'affiche dans la fenêtre de l'éditeur de code. Dans un tel cas, le programme est en mode *arrêt* ; le mode dans lequel le débogage s'effectue.

La boîte de dialogue **Assistant Exception** contient une description de l'erreur, ainsi que des conseils de dépannage qui en suggèrent la cause. Pour plus d'informations, vous pouvez cliquer sur les conseils de dépannage afin d'afficher des rubriques d'aide.

Vous devez résoudre l'erreur pour que le programme puisse continuer ; pour cela, vous devez d'abord inspecter votre code pour en rechercher la cause. Par exemple, si vous soupçonnez qu'une erreur se soit produite parce qu'une variable contient une valeur incorrecte, alors que vous êtes toujours en mode arrêt, vous pouvez utiliser la fonction IntelliSense pour vérifier la valeur de cette variable. Lorsque vous placez le pointeur de la souris sur la variable dans l'éditeur de code, une info-bulle en affiche la valeur. Si la valeur n'est pas celle qui est attendue, revenez au code précédent pour inspecter l'endroit où la valeur a été définie, résolvez ce code, puis poursuivez.

Essayez !

Inspecter la valeur d'une variable

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `RunTimeErrors`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans le gestionnaire d'événements **Form_Load**, ajoutez le code suivant.

   VB

   ```
   Dim miles As Integer = 0
   Dim hours As Integer = 0
   Dim speed As Integer = 0
   ```

   VB

   ```
   miles = 55
   speed = miles / hours
   MsgBox(CStr(speed) & " miles per hour")
   ```
6. Appuyez sur F5 pour exécuter le programme. Une boîte de dialogue **Assistant Exception** accompagne le message "L'exception OverflowException n'a pas été gérée".

   Un trait pointillé reliant la boîte de dialogue à votre fichier de code indique la ligne de code qui a provoqué l'erreur.

   Remarquez que le premier conseil de dépannage de l'**Assistant Exception** suggère de vérifier que vous n'effectuez pas une division par zéro.

7. Placez le pointeur de la souris sur la variable `miles` et maintenez-le quelques secondes sur celle-ci. Une info-bulle doit s'afficher et indiquer "miles 55".
8. Maintenant, placez le pointeur de la souris sur la variable `hours` ; l'info-bulle doit indiquer "hours 0".

   Étant donné que vous ne pouvez pas diviser par zéro et que la valeur de la variable `hours` est nulle, vous identifiez la cause de l'erreur : l'absence de mise à jour de la valeur de `hours`.
9. Ajoutez la ligne de code au-dessus de la ligne `miles = 55`.

   VB

   ```
   hours = 2
   ```
10. Cliquez sur la flèche jaune placée dans la marge gauche du code, puis faites-la glisser jusqu'à la ligne `hours = 2`.

    L'exécution du programme peut alors se poursuivre depuis cette ligne, plutôt que depuis la ligne qui contient l'erreur. C'est indispensable pour que la nouvelle ligne de code que vous venez d'ajouter puisse s'exécuter, afin que votre correctif d'erreur soit reconnu.
11. Appuyez sur F5 pour poursuivre le programme. Une boîte de dialogue s'affiche, avec le message "28 miles per hour".

Étapes suivantes

Dans cette leçon, vous avez appris comment rechercher et résoudre des erreurs d'exécution. Dans la leçon suivante, vous apprendrez le troisième type d'erreurs de programmation : les erreurs logiques. Vous pouvez examiner des techniques de débogage plus avancées dans Présentation détaillée : que se passe-t-il si... je teste du code dans la fenêtre Exécution, puis passer à la leçon sur les erreurs logiques.

Leçon suivante : Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques

Voir aussi

Tâches

Oups, une faute de frappe : recherche et élimination des erreurs du compilateur
Connaissance des bogues : trois types d'erreurs de programmation
Recherche des erreurs : introduction au débogage Visual Basic

Présentation détaillée : que se passe-t-il si... je teste du code dans la fenêtre Exécution

Dans cette leçon, vous apprendrez comment évaluer et exécuter un code à l'aide de la fenêtre **Exécution**.

Dans la leçon précédente, vous avez appris comment corriger des erreurs d'exécution à l'aide de l'**Assistant Exception**. Toutefois, il n'est pas toujours évident de déterminer comment résoudre une erreur et vous pouvez tester un correctif possible sans modifier le code réel. Une fenêtre de débogage spéciale, la fenêtre **Exécution**, vous permet, entre autres possibilités, de procéder de la sorte.

La fenêtre Exécution

Lorsque votre programme est en mode arrêt, la fenêtre **Exécution** peut être utilisée pour exécuter des extraits de code et évaluer des variables ou des expressions. Par exemple, si une erreur d'exécution se produit à cause d'une variable vide, vous pouvez vérifier la valeur d'une variable. Vous pouvez également utiliser la fenêtre **Exécution** pour assigner une valeur à cette variable et vérifier de quelle manière le reste du programme s'exécute en conséquence.

Conseil
Lorsque votre programme s'exécute en mode débogage, vous pouvez mettre votre programme en mode arrêt à tout moment en sélectionnant **Arrêter** dans le menu **Déboguer**.

Vous pouvez exécuter un code dans la fenêtre **Exécution** en l'y entrant comme vous le feriez dans l'éditeur de code, puis en appuyant sur ENTRÉE. Pour évaluer une variable ou une expression, tapez un point d'interrogation suivi de la variable ou de l'expression que vous souhaitez évaluer, puis appuyez sur ENTRÉE ; le résultat est affiché sur la ligne suivante.

Essayez !

Pour tester un code dans la fenêtre Exécution

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Immediate`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Depuis la **Boîte à outils**, faites glisser deux contrôles TextBox et un contrôle **Button** jusqu'au formulaire.
5. Double-cliquez sur le bouton pour ouvrir l'éditeur de code.
6. Dans le gestionnaire d'événements **Button_Click**, ajoutez le code suivant.

   VB

   ```
   Dim miles As Integer = 0
   Dim hours As Integer = 0
   Dim speed As Integer = 0
   ```

   VB

   ```
   miles = CInt(Textbox1.Text)
   hours = CInt(Textbox2.Text)
   speed = miles / hours
   MsgBox(CStr(speed) & " miles per hour")
   ```
7. Appuyez sur F5 pour exécuter le programme. Entrez `100` dans la première zone de texte, puis `0` dans la deuxième.
8. Cliquez sur **Button1**. Le programme s'arrête, et la boîte de dialogue **Assistant Exception** affiche le message "L'exception OverflowException n'a pas été gérée".

9. Dans la fenêtre **Exécution** au bas de l'IDE, entrez `?miles`, puis appuyez sur ENTRÉE.

   La valeur `100` doit figurer sur la ligne suivante.

   > **Conseil**
   >
   > Vous pouvez ouvrir la fenêtre **Exécution** à tout moment en choisissant **Fenêtres**, **Exécution** dans le menu **Déboguer**.

10. Tapez `?hours`, puis appuyez sur ENTRÉE.

    La valeur 0 doit figurer sur la ligne suivante.

11. Tapez `hours = 4` et appuyez sur ENTRÉE. Tapez ensuite `?hours` et appuyez sur ENTRÉE.

    Remarquez que la valeur de `hours` est maintenant de `4`, la valeur que vous avez saisie sur la ligne précédente. Vous pourriez modifier la valeur de `hours` dans la fenêtre **Exécution** sans modifier le code du programme.

12. Appuyez sur F5 pour continuer. Un message s'affiche avec le résultat.

    > **Conseil**
    >
    > Pour empêcher cette erreur d'exécution, ajoutez un gestionnaire d'erreurs qui contrôlera la présence d'un nombre valide dans le bloc **Try**, puis présentez un message à l'utilisateur dans le bloc **Catch**. Pour plus d'informations sur les gestionnaires d'erreurs, consultez Opération à effectuer en cas de problème : gestion des erreurs.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser la fenêtre **Exécution** pour vérifier des valeurs et exécuter un code. Dans la leçon suivante, vous apprendrez comment localiser et corriger des erreurs logiques.

Leçon suivante : Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques

Voir aussi

Tâches

Aïe ! Mon programme n'a pas apprécié ! Recherche et suppression des erreurs d'exécution

Opération à effectuer en cas de problème : gestion des erreurs

Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques

Dans cette leçon, vous apprendrez comment rechercher des erreurs logiques dans votre programme.

Dans les leçons précédentes, vous avez appris comment rechercher et résoudre des erreurs de compilation et d'exécution. Le troisième type d'erreur de programmation, les erreurs logiques, est sans doute le plus difficile de l'identifier. Les erreurs logiques ne génèrent aucun avertissement ; le programme s'exécute, mais vous obtenez des résultats incorrects. Vous devez alors fouiller dans votre code et déterminer pourquoi.

Heureusement, les outils de débogage de Visual Basic sont là pour vous aider. Deux techniques de débogage, définir des *points d'arrêt* et *progresser pas à pas* dans le code, vous aident à inspecter votre code ligne par ligne dans le sens de son exécution pour identifier l'erreur.

Vous pouvez définir un point d'arrêt dans l'**éditeur de code** sur toute ligne de code exécutable. Lorsque le programme s'exécute, les points d'arrêt le forcent à s'arrêter et à passer en mode arrêt une fois cette ligne de code atteinte. Vous pouvez ensuite obtenir les informations souhaitées à propos de l'état du programme à ce moment précis. Vous pouvez alors vérifier la valeur d'une variable, tester des expressions dans la fenêtre **Exécution** ou apporter des modifications à votre code avec la fonctionnalité Modifier & Continuer.

Une fois en mode arrêt, vous pouvez progresser pas à pas dans votre code, en l'exécutant ligne par ligne, afin de pouvoir examiner son fonctionnement. Appuyer sur la touche F8 entraîne l'exécution de la ligne de code courante, puis l'arrêt sur la ligne suivante. Vous pouvez alors inspecter les valeurs des variables pour voir si elles ont été modifiées d'une ligne à l'autre.

Si la ligne de code active appelle une fonction ou une procédure **Sub** présente à un autre endroit dans votre code, l'exécution passe directement à cette procédure lorsque vous appuyez sur F8. Lorsque vous avez exécuté cette procédure pas à pas, vous êtes envoyé à la ligne qui suit celle qui a appelé la procédure. Si vous ne souhaitez pas exécuter la procédure pas à pas, appuyez sur MAJ+F8 pour l'ignorer.

Essayez !

Pour observer une erreur logique

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `LogicErrors`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. À partir de la **Boîte à outils**, faites glisser deux contrôles TextBox et un contrôle Button dans le formulaire.
5. Double-cliquez sur **Button1** pour ouvrir l'éditeur de code.
6. Dans le gestionnaire d'événements **Button1_Click**, ajoutez le code suivant.

   VB

```
Dim minutes As Integer = CInt(Textbox1.Text)
Dim miles As Double = CDbl(Textbox2.Text)
Dim hours As Double = 0
hours = minutes / 60
MsgBox("Average speed " & GetMPH(hours, miles))
```

7. Sous la ligne `End Sub`, ajoutez la fonction suivante :

   VB

```
Function GetMPH(ByVal miles As Double, ByVal hours As Double) _
As String
    GetMPH = CStr(miles / hours)
End Function
```

8. Appuyez sur F5 pour exécuter le programme. Dans la première zone de texte, entrez `10` (pour représenter 10 minutes) et dans la deuxième, entrez `5` (pour représenter 5 miles), puis cliquez sur **Button1**.

   Un message s'affiche et donne la vitesse moyenne : "Average speed 0.03333334". Toutefois, si vous parcourez cinq miles en dix minutes, la réponse correcte est 30 mph.

   Laissez le projet ouvert ; dans la procédure suivante, vous apprendrez comment trouver l'erreur logique.

Identification des erreurs logiques

La logique de programmation du dernier exemple pose de toute évidence un problème. D'après le résultat, vous vous déplacez à moins d'un mile (soit 1 609 m) par heure, et non à trente miles à l'heure comme prévu. Où est l'erreur ?

Dans la procédure suivante, vous définirez un point d'arrêt et progresserez pas à pas dans le code pour localiser l'erreur.

Essayez !

Pour définir un point d'arrêt et progresser pas à pas dans le code

1. Dans l'éditeur de code, recherchez la ligne `hours = minutes / 60`, puis cliquez dans la marge de gauche en regard de cette ligne de code.

   Un point rouge doit s'afficher dans la marge et le code doit être mis en surbrillance en rouge, pour représenter un point d'arrêt.

2. Appuyez sur F5 pour exécuter à nouveau le programme. Dans la première zone de texte, entrez `10` et dans la deuxième zone de texte, entrez `5`. Ensuite, cliquez sur **Button1**.

   Le programme s'arrête lorsqu'il atteint le point d'arrêt. La ligne `hours = minutes / 60` est surlignée en jaune.

   Inspectez les valeurs des variables en maintenant le pointeur de la souris sur celles-ci. La valeur de `hours` doit être `0` et celle de `minutes` `10`.

3. Appuyez sur F8 pour exécuter la ligne `hours = minutes / 60` et passer à la ligne suivante.

   Inspectez les valeurs des variables présentes sur la ligne `MsgBox("Average speed " & GetMPH(hours, miles))` ; la valeur de `hours` doit être maintenant `0.166666672`, et celle de `miles` `5.0`.

4. Appuyez à nouveau sur F8 pour exécuter la ligne active.

   Remarquez que l'exécution descend à la ligne `Function GetMPH`.

   Inspectez les valeurs des variables présentes sur cette ligne. Remarquez que la valeur de `miles` est à présent `0.166666672`, et celle de `hours` à présent `5.0` ; le contraire de ce qu'elles étaient sur la ligne précédente. Vous venez d'identifier l'erreur.

   Laissez le projet ouvert ; dans la procédure suivante, vous apprendrez comment corriger l'erreur logique.

Correction des erreurs logiques

Dans cette dernière procédure, les valeurs des variables `miles` et `hours` ont été interverties. Sauriez-vous dire pourquoi ?

Si vous examinez la ligne `MsgBox("Average speed " & GetMPH(hours, miles))`, vous constatez que les deux arguments, `hours` et `miles`, dans cet ordre, sont passés à la fonction `GetMPH`. Or, si vous examinez la déclaration de fonction `Function GetMPH(ByVal miles As Double, ByVal hours As Double)...`, vous remarquez que les arguments sont répertoriés de la manière suivante : miles en premier (`miles`) et les heures (`hours`) en second.

Une erreur logique s'est produite parce que les arguments sont passés dans le désordre, entraînant ainsi un calcul erroné. Si les arguments avaient été de types différents, une erreur d'exécution se serait produite. Mais ici, ces arguments sont de même type ; aussi, aucune erreur d'exécution n'a lieu. S'il s'agit une erreur simple, le bogue qui en résulte n'en est pas moins difficile à trouver.

Dans la procédure suivante, vous définirez un point d'arrêt et progresserez pas à pas dans le code pour localiser l'erreur.

Essayez !

Pour corriger l'erreur logique

1. Dans l'éditeur de code, modifiez comme suit la ligne `MsgBox("Average speed " & GetMPH(hours, miles))` :

   VB

   ```
   MsgBox("Average speed " & GetMPH(miles, hours))
   ```

2. Effacez le point d'arrêt en cliquant sur le point rouge dans la marge de gauche.

3. Appuyez sur F5 pour exécuter le programme. Dans la première zone de texte, tapez `10` et dans la deuxième `5`. Ensuite, cliquez sur **Button1**.

   Cette fois, le message doit afficher le résultat correct : "Average speed 30".

   Le programme peut sembler corrigé, mais une erreur logique encore plus difficile à identifier est encore présente. Si vous souhaitez la rechercher, laissez le projet ouvert ; vous l'utiliserez à nouveau dans la leçon Crédit supplémentaire : quelque chose ne fonctionne toujours pas.

Étapes suivantes

Dans cette leçon, vous avez appris comment rechercher et corriger une erreur logique. À ce stade, vous pouvez passer à la leçon suivante sur l'utilisation des commentaires, ou vous essayer à la recherche d'une autre erreur logique dans Crédit supplémentaire : quelque chose ne fonctionne toujours pas.

Leçon suivante : Prise de notes dans vos programmes : utilisation des commentaires

Voir aussi

Tâches

Aïe ! Mon programme n'a pas apprécié ! Recherche et suppression des erreurs d'exécution
Connaissance des bogues : trois types d'erreurs de programmation
Recherche des erreurs : introduction au débogage Visual Basic

Crédit supplémentaire : quelque chose ne fonctionne toujours pas

Dans cette leçon, vous apprendrez à localiser une erreur logique qui ne se produit que dans certaines situations.

Dans la leçon précédente, Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques, vous avez appris comment rechercher et corriger une erreur dans la logique. L'exemple de code de cette leçon dissimule encore un bogue malicieux plus difficile à trouver, car ces effets ne se produisent que dans certaines situations.

Test d'un programme

En tant que programmeur, lorsqu'il s'agit de tester votre programme pour savoir s'il se comporte comme vous le souhaitez, vous partez avec un handicap. Vous savez comment il doit fonctionner ; aussi est-il improbable que vous fassiez une erreur susceptible de dévoiler une erreur logique. En revanche, un utilisateur peu familiarisé avec le programme peut et va effectuer des opérations auxquelles vous n'auriez pas pensé.

Par exemple, dans un programme qui calcule une vitesse en miles par heure en divisant le nombre de miles parcourus par le nombre d'heures écoulées lors du trajet, que se passe-t-il si l'utilisateur entre la valeur zéro pour les heures ou les miles ? Essayons pour voir.

Essayez !

Pour tester le programme

1. Ouvrez le projet `LogicErrors` créé dans la leçon précédente,
   Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques.

Remarque
Si vous n'avez pas terminé ou enregistré le projet précédent, vous devez revenir en arrière et l'achever pour pouvoir poursuivre.

2. Appuyez sur F5 pour exécuter le programme. Dans la première zone de texte, entrez `0` (pour représenter les minutes) et dans la deuxième, entrez `5` (pour représenter cinq miles), puis cliquez sur **Button1**.

   Un message s'affiche et indique une vitesse moyenne infinie : "Average speed Infinity".

   Laissez le projet ouvert ; dans la procédure suivante, vous apprendrez comment trouver l'erreur logique.

5 divisé par 0 = l'infini ?

Dans la procédure précédente, l'infini peut ne pas être ce que vous attendiez, pourtant sa valeur est mathématiquement correcte : 5 contient un nombre infini de fois la valeur 0. Il n'en demeure pas moins que ce n'est pas le résultat que vous souhaitez présenter aux utilisateurs de votre programme. Voyez-vous un moyen d'empêcher cette situation ?

Vous pouvez envisager l'ajout d'un gestionnaire d'erreurs (procédure que vous avez apprise dans la leçon Opération à effectuer en cas de problème : gestion des erreurs). Toutefois, cela ne fonctionnerait pas dans ce cas-ci. Le résultat « infini » n'est pas une erreur ; ce n'est simplement pas ce que vous souhaitez obtenir.

Sachant que l'affichage d'une vitesse nulle est vraisemblablement inutile, vous pouvez résoudre le problème en appliquant un test à la valeur zéro, puis en avertissant l'utilisateur qu'il doit saisir une valeur supérieure. Tant que vous y êtes, vous pouvez également empêcher l'utilisateur d'entrer des nombres négatifs, car ceux-ci peuvent également produire un résultat erroné.

Dans la procédure suivante, vous modifierez le code du gestionnaire d'événements **Button1_Click** pour appeler la fonction `GetMPH` uniquement lorsque les valeurs sont supérieures à zéro.

Essayez !

Pour corriger le bogue

1. Dans l'éditeur de code, modifiez le code dans le gestionnaire d'événements **Button1_Click** comme suit :

   VB

```
Dim minutes As Integer = CInt(Textbox1.Text)
Dim miles As Double = CDbl(Textbox2.Text)
Dim hours As Double = 0
If minutes <= 0 Or miles <= 0 Then
  MsgBox("Please enter a number greater than zero")
Else
  hours = minutes / 60
  MsgBox("Average speed " & GetMPH(miles, hours))
End If
```

2. Appuyez sur F5 pour exécuter à nouveau le programme. Dans la première zone de texte, entrez `0`, et dans la deuxième zone de texte, entrez `5`. Ensuite, cliquez sur **Button1**.

   Le message qui vous invite à entrer un nombre supérieur à zéro doit s'afficher. Essayez de tester votre programme avec d'autres combinaisons de nombres jusqu'à ce que vous soyez satisfaits de la correction de l'erreur.

Étapes suivantes

Dans cette leçon, vous avez appris comment rechercher et résoudre une erreur logique qui provoque un comportement inattendu. Dans la leçon suivante, vous apprendrez comment utiliser des commentaires dans votre code.

Leçon suivante : Prise de notes dans vos programmes : utilisation des commentaires

Voir aussi

Tâches

Comment ? Je n'étais pas censé faire cela ! Suivi des erreurs logiques
Opération à effectuer en cas de problème : gestion des erreurs
Connaissance des bogues : trois types d'erreurs de programmation
Recherche des erreurs : introduction au débogage Visual Basic

Référence

/, opérateur (Visual Basic)

Prise de notes dans vos programmes : utilisation des commentaires

Dans cette leçon, vous apprendrez comment placer des commentaires dans le code de vos programmes.

Le code qui compose un programme peut être difficile à lire et à comprendre, en particulier si vous n'en êtes pas l'auteur d'origine. En utilisant des *commentaires*, vous pouvez créer des remarques destinées à vous-même ou à d'autres utilisateurs de votre code.

Les commentaires sont des entrées de texte saisies dans l'éditeur de code. Le compilateur Visual Basic les ignore lorsque le programme s'exécute Vous pouvez donc vous adresser une remarque qui explique la fonction d'une section de code particulière de votre programme, ou qui vous rappelle d'éventuelles tâches de programmation à terminer.

Pour créer un commentaire, débutez une ligne par le caractère `'`. L'exemple suivant montre comment utiliser un commentaire.

VB

```
' This is a comment. WOW!
```

Vous pouvez également ajouter des commentaires à la fin des lignes en utilisant le caractère `'`. Cette utilisation est fréquente pour fournir des commentaires relatifs à des lignes de code individuelles, comme l'illustre l'exemple suivant.

VB

```
MsgBox("Hello World!") ' This line causes a message box to appear.
```

Comme dans le cas de commentaires qui occupent leur propre ligne, le programme ignore, ici aussi, tout ce qui suit le caractère `'` sur la ligne.

Utilisation des commentaires pour le débogage

Une autre utilisation répandue des commentaires consiste à empêcher temporairement une ligne de code de s'exécuter pendant que vous déboguez votre programme. Prenons l'exemple d'une ligne de code affichant un message.

VB

```
MsgBox("Hello World!")
```

Si vous souhaitez exécuter le programme sans afficher ce message, mais sans pour autant supprimer la ligne de manière définitive, utilisez le caractère de commentaire (') pour la dissimuler temporairement à votre programme, comme illustré ci-dessous.

VB

```
' MsgBox("Hello World!")
```

Etant donné que tout ce qui suit le caractère `'` est ignoré, le programme s'exécute sans exécuter cette ligne. Pour afficher à nouveau le message, il vous suffit de supprimer le caractère `'`.

Essayez !

Pour insérer des commentaires

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Comments`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.

4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans le gestionnaire d'événements **Form1_Load**, tapez le code suivant.

   VB

   ```
   ' This code will cause two message boxes to appear
   MsgBox("This is Message Box 1") ' Display Message Box 1
   MsgBox("This is Message Box 2") ' Display Message Box 2
   ```

6. Appuyez sur F5 pour exécuter le programme.

   Le programme démarre et les deux messages s'affichent tour à tour.
7. Dans le menu **Déboguer**, choisissez **Arrêter le débogage** pour terminer le programme.
8. Dans l'éditeur de code, ajoutez un caractère de commentaire (') à la première ligne du message, comme suit.

   VB

   ```
   ' MsgBox("This is MessageBox 1") ' Ignore Message Box 1
   ```

9. Appuyez sur F5 pour exécuter le programme.

   Remarquez que, cette fois, la première ligne de message est ignorée par le programme, et que seul le deuxième message s'affiche.

Étapes suivantes

Dans cette leçon, vous avez appris à utiliser des commentaires pour placer des remarques dans votre code et désactiver individuellement et temporairement des lignes de code. Cette leçon termine la série sur le débogage. Dans la série de leçons suivante, vous apprendrez comment utiliser une base de données pour enregistrer et extraire des informations pour votre programme.

Leçon suivante : Gestion de vos enregistrements : utilisation de données dans votre programme

Voir aussi

Concepts

Commentaires dans le code

Autres ressources

Qu'est-ce qui s'est mal passé ? Recherche et résolution des erreurs à l'aide du débogage

Gestion de vos enregistrements : utilisation de données dans votre programme

La plupart des programmes utilisent des données d'une façon ou d'une autre. Par exemple, dans une leçon précédente, vous avez saisi des données sous forme de nombres ; ces données ont ensuite été utilisées dans un calcul et le résultat retourné dans un message.

Dans des programmes très simples, les données sont représentées sous la forme de champs à même les programmes. Toutefois, dans le cas de programmes plus complexes, les données sont enregistrées dans une structure séparée du programme, appelée *base de données*.

Dans cet ensemble de leçons, vous apprendrez comment créer une base de données et l'utiliser pour afficher et mettre à jour des données dans vos programmes.

Dans cette section

Stockage et accès aux données

Création de votre première base de données

Obtention des informations requises : connexion à une base de données existante

Affichage d'informations destinées à l'utilisateur : affichage des données dans votre interface utilisateur

Ajout ou modification de vos enregistrements : mise à jour de données

Rubriques connexes

Visite guidée de Visual Basic

Stockage et accès aux données

Dans cette leçon, vous apprendrez à utiliser une base de données pour stocker des données et y accéder.

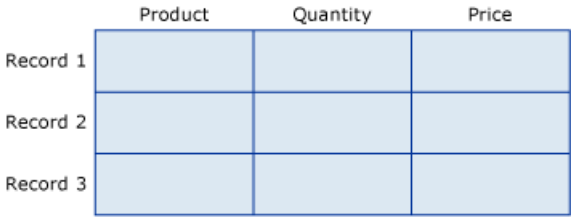
Les *données* sont un concept central de la programmation informatique. La plupart des programmes utilisent des données d'une façon ou d'une autre. Par exemple, dans une leçon précédente, vous avez saisi des données sous forme de nombres ; ces données ont ensuite été utilisées dans un calcul et retournées dans un message.

Dans des programmes très simples, les données sont représentées sous la forme de champs. Toutefois, dans le cas de programmes plus complexes, les données sont enregistrées dans une structure séparée du programme, appelée *base de données*.

Qu'est-ce qu'une base de données ?

Une base de données est une collection de données, enregistrée dans un fichier indépendant de votre programme. Les données enregistrées dans une base de données peuvent présenter de nombreux types distincts, notamment du texte, des nombres ou encore des images. Plusieurs programmes peuvent se connecter à la même base de données pour afficher et mettre à jour les données qu'elle contient.

Une base de données est généralement divisée en une ou plusieurs *tables*. Une table est une collection d'enregistrements connexes. Par exemple, si vous utilisez une base de données qui contient les données d'une PME, une de ses tables peut consigner les produits, une autre les commandes, et une autre les clients.



Chaque table est organisée selon une grille de colonnes et de lignes. Les colonnes représentent des catégories pour les données présentes dans un enregistrement, et les lignes, les enregistrements. Par exemple, dans l'illustration précitée, la table **Commandes** contient une ligne (un enregistrement) distincte pour chaque commande, et des colonnes qui représentent le produit commandé, ainsi que sa quantité et son prix.

Débuter avec les données

Pour accéder aux données d'une base de données depuis votre programme, vous devez, en premier lieu, disposer d'une base de données à laquelle accéder. Visual Basic vous facilite la création de votre propre base de données, ainsi que l'utilisation d'une base de données créée par un tiers.

Visual Basic Express permet d'accéder à deux types de bases de données différents : les bases de données Microsoft SQL Server et les bases de données Microsoft Access. Dans le cadre des présentes leçons, vous utiliserez une base de données SQL Server.

Une fois que vous disposez d'une base de données, vous la connectez à votre programme à l'aide d'un objet appelé *DataSet*, puis vous connectez les champs ou les contrôles présents dans un formulaire aux données présentes dans la base de données, à l'aide d'une technique appelée *liaison de données*.

Lorsqu'un champ de votre programme, par exemple un contrôle **TextBox** , est *lié* à une colonne d'une table de base de données, vous pouvez afficher les données de cette colonne dans la zone de texte, les y modifier, puis les enregistrer dans la base de données, ou encore entrer les données d'un nouvel enregistrement et les ajouter à la base de données.

Même si cela peut paraître compliqué, ce n'est réellement pas le cas. Les outils de base de données de Visual Basic font de l'exploitation de données un jeu d'enfant, comme vous le verrez dans les leçons suivantes.

Étapes suivantes

Dans cette leçon, vous avez appris ce qu'est une base de données et comment elle se lie avec votre programme. Dans la leçon suivante, vous apprendrez comment créer une base de données utilisable par vos programmes.

Remarque
Pour créer une base de données SQL Server et y accéder avec Visual Basic Express, vous devez également installer SQL Server Express. Cette application est installée par défaut lorsque vous installez Visual Basic Express. Toutefois, si vous avez choisi de ne le pas l'installer, vous devrez le faire avant de passer à la leçon suivante.

Leçon suivante : Création de votre première base de données

Voir aussi

Autres ressources

Gestion de vos enregistrements : utilisation de données dans votre programme

Visite guidée de Visual Basic

Création de votre première base de données

Dans cette leçon, vous apprendrez comment créer une base de données que vous utiliserez dans les leçons suivantes pour créer un programme de carnet d'adresses.

Dans la leçon précédente, vous avez appris qu'une base de données permet de stocker et de récupérer des données pour vos programmes Visual Basic. En premier lieu, vous devez disposer d'une base de données à laquelle accéder. Alors que vous pourriez utiliser une base de données existante, dans le cadre des présentes leçons, vous apprendrez comment créer une base de données à l'aide de Visual Database Tools, une application intégrée à Visual Basic.

Composants requis

Pour créer une base de données SQL Server et y accéder avec Visual Basic Express, vous devez également installer SQL Server Express. Cette application est installée par défaut lorsque vous installez Visual Basic Express. Toutefois, si vous avez choisi de ne le pas l'installer, vous devrez le faire avant de poursuivre.

Essayez !

Pour créer une base de données

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `FirstDatabase`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.
5. Dans la boîte de dialogue **Ajouter un nouvel élément**, sélectionnez **Base de données SQL**.
6. Dans la zone **Nom**, tapez `FirstDatabase`, puis cliquez sur **Ajouter**.

   L'Assistant **Configuration de source de données** démarre.
7. Dans l'Assistant **Configuration de source de données**, cliquez sur **Annuler**.

   Une nouvelle base de données, `FirstDatabase.mdf`, est ajoutée au projet et s'affiche dans l'**Explorateur de solutions**.

Ajout d'une table

Comme vous pouvez le voir dans la procédure ci-dessus, créer une base de données constitue une opération facile. À ce stade, la base de données n'est pas utile, car elle ne contient aucune donnée. Dans la procédure suivante, vous ajouterez une table à la base de données ; dans le cas présent, une table pour stocker des adresses.

Essayez !

Pour ajouter une table à la base de données

1. Dans le menu **Affichage**, sélectionnez **Explorateur de bases de données**.
2. Dans l'**Explorateur de bases de données**, développez le nœud **FirstDatabase.mdf** (cliquez sur le signe plus) et sélectionnez le nœud **Tables**.
3. Dans le menu **Données**, choisissez **Ajouter une nouvelle table**.

   Une fenêtre du **Concepteur de tables** s'ouvre.
4. Dans la fenêtre **Propriétés**, sélectionnez **Nom** et tapez `Addresses`.
5. Dans la fenêtre du **Concepteur de tables**, sélectionnez le champ **Nom de la colonne** et tapez `FirstName`.
6. Sélectionnez le champ **Type de données**, puis **nvarchar(50)** dans la liste déroulante. La colonne **AllowNulls** est automatiquement activée.

   Vous avez défini la première colonne de votre nouvelle table.

   Répétez les deux étapes précédentes pour ajouter quatre colonnes de plus dotées des valeurs suivantes :

a. **Nom de la colonne** : `LastName`, **Type de données**: **nvarchar (50)**

b. **Nom de la colonne** : `StreetAddress`, **Type de données**: **nvarchar (50)**

c. **Nom de la colonne** : `City`, **Type de données**: **nvarchar (50)**

d. **Nom de la colonne** : `Phone`, **Type de données**: **nvarchar (50)**

7. Dans le menu **Fichier**, sélectionnez **Enregistrer les adresses**.

Ajout d'une clé

Votre base de données dispose à présent d'une table. Vous pouvez l'utiliser pour stocker des noms, des adresses et des numéros de téléphone destinés à votre carnet d'adresses. Il reste encore une étape : l'ajout d'une *clé* afin d'empêcher tout enregistrement en doublon.

Une colonne clé, également connue sous le nom de *clé primaire*, désigne une ou plusieurs colonnes de votre table en tant que valeur unique. La table qui contient cette valeur ne peut contenir qu'une seule ligne ; si vous tentez d'entrer une deuxième ligne de la même valeur, une erreur s'affiche.

Dans le cas de la table `Addresses`, désignez à la fois les colonnes `FirstName` et `LastName` en tant que clé primaire ; en effet, s'il est possible que plusieurs personnes de votre connaissance aient le même prénom ou le même nom, il est peu probable que vous connaissiez des personnes ayant à la fois le même nom et le même prénom.

Essayez !

Pour ajouter une clé à la table

1. Dans le **Concepteur de tables**, désactivez la case à cocher **Null autorisé** pour les lignes **FirstName** et **LastName**
2. Sélectionnez les lignes **FirstName** et **LastName**.

Conseil
Pour sélectionner ces deux éléments, cliquez sur le carré gris placé à gauche du champ **FirstName**, puis appuyez sur la touche CTRL tandis que vous cliquez sur la ligne **LastName**.

3. Dans le menu **Concepteur de tables**, cliquez sur **Définir la clé primaire**.

   Un petit symbole clé s'affiche à gauche de chaque ligne.

4. Dans le menu **Fichier**, sélectionnez **Enregistrer les adresses**.

Ajout de données

Vous disposez maintenant d'une base de données qui contient une table unique `Addresses`. Bien sûr, sans données, une base de données n'est pas très utile. Dans la procédure suivante, vous ajouterez des données à la table `Addresses`. Si vous le souhaitez, vous pouvez remplacer les noms et adresses fournis dans l'exemple par ceux de personnes que vous connaissez.

Essayez !

Pour ajouter des données à la table

1. Dans l'**Explorateur de bases de données**, développez le nœud **Tables** et sélectionnez le nœud **Addresses**, puis dans le menu **Données**, choisissez **Afficher les données de la table**.

   Une fenêtre de table de données s'ouvre.

2. Dans la fenêtre de table de données, sélectionnez le champ **FirstName**, puis tapez `Samantha`.

Remarque
Remarquez que lors de la première sélection du champ, la valeur *NULL* s'affiche dans chaque champ ; NULL est un ter me spécifique aux bases de données qui indique que le champ est vide.

3. Sélectionnez le champ **LastName**, puis tapez `Smith`.
4. Sélectionnez le champ **StreeAddress**, puis tapez `123 45th Ave. E`.

5. Sélectionnez le champ **City**, puis tapez `Seattle`.
6. Sélectionnez le champ **Phone**, puis tapez `2065550100`. Ensuite, appuyez sur la touche TAB.

   Vous venez de définir le premier enregistrement de la table `Addresses`.

   Répétez les cinq étapes précédentes pour ajouter deux enregistrements de plus dotés des valeurs suivantes :

   a. **FirstName** : `Michael`, **LastName** : `Alexander`, **StreetAddress** : `789 W. Capital Way`, **City** : `Tacoma`, **Phone** : `2065550101`.

   b. **FirstName** : `Andrea`, **LastName** : `Dunker`, **StreetAddress** : `722 Moss Bay Blvd`, **City** : `Kirkland`, **Phone** : `2065550102`.
7. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer le projet et la base de données.

Lorsque vous entriez les données, vous avez pu remarquer qu'une petite icône en forme de crayon est apparue à côté de celles-ci ; lorsque vous avez utilisé la touche TAB pour passer à la ligne suivante, ces icônes ont disparu. L'icône en forme de crayon signifie que les données n'ont pas été enregistrées dans la base de données. Lorsque vous vous éloignez de la ligne dans laquelle vous avez entré des données, les données de la ligne entière sont enregistrées automatiquement dans la base de données.

Étapes suivantes

Dans cette leçon, vous avez créé une base de données, ajouté une table de base de données, puis ajouté des enregistrements à la table dans l'IDE. Dans la leçon suivante, vous apprendrez comment utiliser une base de données dans votre programme.

Leçon suivante : Obtention des informations requises : connexion à une base de données existante.

Voir aussi

Tâches

Stockage et accès aux données

Autres ressources

Gestion de vos enregistrements : utilisation de données dans votre programme

Visite guidée de Visual Basic

Obtention des informations requises : connexion à une base de données existante

Dans cette leçon, vous apprendrez comment connecter votre programme à une base de données existante.

Se connecter à une base de données existante est facile. Vous pouvez utiliser les outils visuels de Visual Basic Express pour naviguer jusqu'à votre base de données et ajouter une copie locale à votre projet. Dans cette leçon, vous créerez un projet et le connecterez à la base de données `Addresses` que vous avez créée dans la leçon précédente.

Essayez !

Pour se connecter à une base de données existante

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Addresses`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Dans l'**Explorateur de solutions**, sélectionnez le nœud `FirstDatabase.mdf`.
5. Dans la fenêtre **Propriétés**, sélectionnez la propriété **Copier dans le répertoire de sortie** et modifiez la valeur en **Copier si plus récent**.
6. Dans l'**Explorateur de solutions**, cliquez sur l'onglet **Sources de données**.
7. Dans la fenêtre **Sources de données**, cliquez sur **Ajouter une nouvelle source de données**.

   L'**Assistant Configuration de source de données** s'affiche.
8. Sélectionnez **Base de données** et cliquez sur **Suivant**.
9. Cliquez sur le bouton **Nouvelle connexion**.

   La boîte de dialogue **Ajouter une connexion** s'ouvre.
10. Dans la boîte de dialogue **Ajouter une connexion**, si la **Source de données** n'est pas **Fichier de base de données Microsoft SQL Server (SQL Client)**, cliquez sur le bouton **Modifier** et sélectionnez **Fichier de base de données Microsoft SQL Server** dans la boîte de dialogue **Modifier la source de données**. Cliquez sur **OK**.
11. Cliquez sur le bouton **Parcourir**, naviguez jusqu'à l'emplacement où vous avez enregistré la base de données `FirstDatabase.mdf`, puis cliquez sur **Ouvrir**.
12. Cliquez sur **OK** pour fermer la boîte de dialogue, puis dans l'**Assistant Configuration de source de données**, cliquez sur **Suivant**. Si vous êtes invité à copier le fichier de données dans votre projet, cliquez sur **Oui**.
13. Sur la page suivante de l'Assistant, la case à cocher **Oui, enregistrer la connexion en tant que** doit être activée. Cliquez sur **Suivant** pour continuer.
14. Sur la page **Choisir vos objets de base de données**, développez le nœud **Tables**, puis activez la case à cocher correspondant à la table **Addresses**.
15. Cliquez sur **Terminer** pour terminer.

    Un fichier de base de données local est ajouté à votre projet. Remarquez qu'un objet `FirstDatabaseDataSet` a été ajouté à la fenêtre **Sources de données**.
16. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre projet.

Étapes suivantes

Dans cette leçon, vous avez appris comment ajouter une base de données locale à votre projet. Dans la leçon suivante, vous apprendrez comment créer une interface utilisateur pour afficher les enregistrements présents dans votre base de données.

Leçon suivante : Affichage d'informations destinées à l'utilisateur : affichage des données dans votre interface utilisateur

Voir aussi

Tâches

Création de votre première base de données

Autres ressources

Gestion de vos enregistrements : utilisation de données dans votre programme

Visite guidée de Visual Basic

Affichage d'informations destinées à l'utilisateur : affichage des données dans votre interface utilisateur

Dans cette leçon, vous apprendrez comment créer une interface utilisateur de base pour consulter les données enregistrées dans votre base de données locale.

Maintenant que vous avez créé une connexion à un fichier de base de données local, l'étape suivante consiste à créer une interface utilisateur pour afficher les données. La programmation qui sous-tend la récupération de données d'une base de données et leur affichage dans une interface utilisateur est très compliquée. Heureusement, Visual Basic crée et configure automatiquement pour vous les objets de donnée requis ; tout ce que vous avez à faire se résume donc à sélectionner et positionner ces objets. Dans cette leçon, vous apprendrez comment créer un formulaire d'affichage de données simple.

Essayez !

Pour créer un formulaire d'accès aux données

1. Ouvrez le projet `Addresses` de la leçon précédente. Si vous ne l'avez pas fini, allez à Obtention des informations requises : connexion à une base de données existante et terminez la leçon avant de poursuivre.
2. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
3. Dans l'**Explorateur de solutions**, cliquez sur l'onglet **Sources de données**.

   Dans la fenêtre **Sources de données**, prenez le temps d'examiner les nœuds **FirstDatabaseDataSet** et **Addresses**. Vous pouvez développer le nœud **Addresses** pour afficher les différents champs de la table.
4. Faites glisser le nœud **Addresses** depuis la fenêtre **Sources de données** jusqu'au formulaire.

   Remarque

   Certains contrôles sont ajoutés automatiquement au formulaire. Plusieurs composants sont créés et ajoutés à la barre d'état des composants, sous le formulaire. Un contrôle **DataGridView** affiche les lignes et les colonnes de la table ; un autre sert à la navigation (**AddressesBindingNavigator**). En outre, Visual Basic crée des composants qui se connecten t à la base de données, gèrent la récupération et la mise à jour des données, et enregistrent ces dernières dans un **Data Set** local (respectivement **AddressesBindingSource**, **AddressesTableAdapter** et **FirstDatabaseDataSet**)
5. Sélectionnez le contrôle **AddressesDataGridView**, et dans la fenêtre **Propriétés**, affectez à la propriété **Dock** la valeur **Fill** (cliquez sur le bouton central).

   La grille se développe pour remplir le formulaire.
6. Appuyez sur F5 pour exécuter le programme.

   Les données issues de la table `Addresses` sont affichées dans le contrôle **DataGridView** sur le formulaire. Vous pouvez utiliser les contrôles du **BindingNavigator** pour naviguer au sein des lignes, et même pour ajouter ou supprimer des enregistrements. Vous pouvez également apporter des modifications aux enregistrements en modifiant les données affichées dans la grille. Toutefois, ces modifications ne seront pas enregistrées, à moins que vous cliquiez sur l'icône **Enregistrer les données**. Dans la rubrique suivante, vous apprendrez comment enregistrer automatiquement les modifications apportées à vos données.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser les outils visuels Visual Basic pour créer une interface utilisateur de base orientée données. Dans la leçon suivante, vous apprendrez comment créer un formulaire de saisie de données et comment enregistrer les modifications apportées à votre base de données.

Leçon suivante : Ajout ou modification de vos enregistrements : mise à jour de données

Voir aussi

Tâches

Obtention des informations requises : connexion à une base de données existante

Autres ressources

Gestion de vos enregistrements : utilisation de données dans votre programme
Visite guidée de Visual Basic

Ajout ou modification de vos enregistrements : mise à jour de données

Dans cette leçon, vous apprendrez comment créer un formulaire de saisie de données pour mettre à jour des données dans votre base de données locale.

Dans les trois dernières leçons, vous avez créé une base de données, ajouté un fichier de base de données à votre projet, et créé une interface utilisateur de base. Comme vous avez pu le remarquer, vous pouvez apporter des modifications aux adresses et même ajouter de nouveaux enregistrements. Toutefois, si vous fermez le programme et le redémarrez, ces modifications sont perdues.

Ces données étaient, en fait, des copies des données présentes dans la base de données, enregistrées dans un **DataSet** local. À chaque démarrage du programme, le **DataSet** récupère ses données de la base de données. Lorsque des modifications sont apportées au **DataSet**, elles ne sont pas répercutées dans la base de données.

Si vous cliquez sur le bouton **Enregistrer** sur le contrôle **AddressesBindingNavigator**, toutes les modifications sont copiées en retour du **DataSet** dans la base de données. Sachant qu'un utilisateur ne pense pas systématiquement à enregistrer son travail, ajoutez un code pour enregistrer automatiquement les modifications effectuées dans la base de données lors de la fermeture du programme. Tant que vous y êtes, modifierez également l'interface utilisateur pour faciliter la saisie des données.

Essayez !

Pour mettre à jour votre fichier de base de données local

1. Ouvrez le projet `Addresses` de la leçon précédente. Si vous n'avez pas encore terminé la leçon précédente, allez à Affichage d'informations destinées à l'utilisateur : affichage des données dans votre interface utilisateur et terminez les étapes nécessaires.
2. Dans l'**Explorateur de solutions**, sélectionnez **Form1**, puis dans le menu **Affichage**, choisissez **Concepteur**.
3. Dans le formulaire, sélectionnez le contrôle **AddressesDatGridView** et supprimez-le.
4. Dans l'**Explorateur de solutions**, cliquez sur l'onglet **Sources de données**.
5. Dans la fenêtre **Sources de données**, sélectionnez le tableau **Addresses**, puis sélectionnez **Détails** dans la liste déroulante.
6. Faites glisser le nœud **Addresses** de la fenêtre **Sources de données** dans le formulaire.

   Des contrôles **TextBox** sont ajoutés pour chaque champ de la table, ainsi que des contrôles **Label** qui décrivent les champs.
7. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
8. Dans la liste déroulante **Événements**, cliquez sur **FormClosing**.
9. Dans le gestionnaire d'événements **Form1_FormClosing**, tapez le code suivant :

   ```
   Me.AddressesBindingSource.EndEdit()
   Me.AddressesTableAdapter.Update(Me.FirstDatabaseDataSet.Addresses)
   ```

   Ce code contraint le **AddressesTableAdapter** à copier toute modification apportée au groupe de données dans la base de données locale.
10. Appuyez sur F5 pour exécuter le programme.

    Apportez des modifications à certaines données ou ajoutez un nouvel enregistrement, puis fermez le formulaire.
11. Appuyez à nouveau sur F5. Vos modifications doivent être enregistrées.

Dans cet ensemble de leçons, vous avez appris comment créer une base de données et un programme pour y accéder. Dans l'ensemble de leçons suivant, vous étudierez les classes et les plans des objets que vous pouvez réutiliser dans vos programmes.

Leçon suivante : Programmation avec des objets : utilisation de classes

Voir aussi

Tâches

Obtention des informations requises : connexion à une base de données existante

Autres ressources

Gestion de vos enregistrements : utilisation de données dans votre programme

Visite guidée de Visual Basic

Programmation avec des objets : utilisation de classes

Comme vous l'avez appris dans une leçon précédente, les programmes Visual Basic sont construits au moyen d'objets tels que des formulaires et des contrôles. Les objets peuvent également représenter des éléments du monde réel, notamment une personne ou un ordinateur, voire un élément plus abstrait, tel qu'un compte bancaire.

Une *classe* est simplement la représentation d'un type d'objet ; considérez la classe comme le plan d'un objet. Tout comme un même plan peut être employé pour construire plusieurs bâtiments, une même classe peut servir à créer plusieurs exemplaires d'un objet.

Dans les leçons suivantes, vous apprendrez comment utiliser des classes dans vos programmes Visual Basic.

Dans cette section

Qu'est-ce qu'une classe ?

Modélisation d'un objet réel : création de votre première classe

Ajout de propriétés à votre classe

Ajout de méthodes à votre classe

Test de votre classe

Génération de votre classe sur une classe existante : utilisation de l'héritage

Suivi d'éléments : utilisation de collections pour gérer plusieurs objets

Rubriques connexes

Présentation détaillée : création de plusieurs versions de la même méthode avec surcharge

Présentation détaillée : substitution de membres

Présentation détaillée : utilisation d'une boucle For Each...Next dans une collection

Qu'est-ce qu'une classe ?

Dans cette leçon, vous apprendrez comment utiliser des classes pour représenter des objets dans vos programmes.

Comme vous l'avez appris dans une leçon précédente, les programmes Visual Basic sont construits au moyen d'objets tels que des formulaires ou des contrôles. Les objets peuvent également représenter des éléments du monde réel, notamment une personne, un ordinateur, voire un élément plus abstrait, tel qu'un compte bancaire.

Une *classe* est simplement une représentation d'un type d'objet ; considérez-la comme un plan qui décrit l'objet. Tout comme un même plan peut être employé pour construire plusieurs bâtiments, une même classe peut servir à créer plusieurs exemplaires d'un objet.

Sans vous en rendre compte, vous avez déjà utilisé des classes. Par exemple, le contrôle **TextBox** est défini par une classe **TextBox** qui définit son aspect et ses fonctions. Chaque fois que vous faites glisser un contrôle **TextBox** sur un formulaire, vous créez en fait une nouvelle *instance* de la classe **TextBox**.

Chaque contrôle **TextBox** est une copie exacte mais distincte de la classe qui le définit ; la classe TextBox. Dans la mesure où chaque objet constitue une "instance" distincte d'une classe, l'action qui consiste à créer une classe se nomme *instanciation*.

Jusqu'à maintenant, vous avez ajouté des contrôles **TextBox** à votre formulaire en les faisant glisser depuis la **Boîte à outils**. Mais vous pouvez aussi instancier un objet **TextBox** dans votre code en le déclarant avec le mot clé **New**.

VB

```
Dim Textbox1 As New TextBox
```

Vous en saurez plus sur la création et l'utilisation des classes dans les prochaines leçons.

Que contient une classe ?

Dans une leçon précédente, Présentation détaillée : explication des propriétés, des méthodes et des événements, vous avez appris que tous les objets ont des propriétés qui décrivent leurs attributs, des méthodes qui définissent leurs actions, et des événements qui définissent leurs réactions. De la même manière, la classe qui définit un objet dispose de ses propres propriétés, méthodes et événements (parfois appelés *membres*), qui sont transmis à toutes les instances de cette classe.

Par exemple, une classe qui représente un compte bancaire peut disposer de propriétés telles que `AccountNumber` ou `AccountBalance`, de méthodes telles que `CalculateInterest`, et d'événements tels que `BalanceChanged`. Une fois un objet de compte bancaire instancié, vous avez accès à ses propriétés, méthodes et événements, tout comme ce serait le cas avec un objet tel que **TextBox**.

Certains membres d'une classe sont privés : ils ne sont accessibles qu'au code présent dans la classe. Par exemple, une classe de compte bancaire peut disposer d'une méthode pour calculer un solde. Vous autoriseriez un programme à lire ce solde, mais vous ne souhaiteriez pas qu'il soit en mesure de le modifier directement.

Vous pouvez masquer certains membres d'une classe en les déclarant comme étant **Private**. Vous pouvez, à l'inverse, les exposer en les déclarant comme étant **Public**. Vous pouvez également autoriser l'accès à une propriété, mais empêcher le programme d'en modifier la valeur en le déclarant comme étant **ReadOnly**. Le code suivant illustre ce à quoi peut ressembler une classe `BankAccount`.

VB

```
Class BankAccount
    Private AccountNumber As String
    Private AccountBalance As Decimal
    Public Sub UpdateBalance()
        ' add code to recalculate balance.
    End Sub
    ReadOnly Property Balance() As Decimal
        Get
            Return AccountBalance
        End Get
    End Property
End Class
```

Étapes suivantes

Dans cette rubrique, vous avez appris les notions de base des classes, ainsi que certains nouveaux éléments de terminologie. Dans la leçon suivante, vous apprendrez comment créer une classe.

Leçon suivante : Modélisation d'un objet réel : création de votre première classe.

Voir aussi

Concepts

Présentation détaillée : explication des propriétés, des méthodes et des événements

Classes : modèles d'objets

Autres ressources

Programmation avec des objets : utilisation de classes

Modélisation d'un objet réel : création de votre première classe

Dans cette leçon, vous apprendrez comment créer une classe à l'aide d'un projet **Bibliothèque de classes**.

Dans la leçon précédente, vous avez appris que les classes peuvent être utilisées en tant que plans pour des objets qui modélisent des éléments du monde réel. Une des meilleures raisons d'utiliser une classe tient dans le fait qu'une fois celle-ci créée pour un certain type d'objet, vous pouvez la réutiliser dans n'importe quel projet.

Par exemple, nombre de programmes que vous développez peuvent impliquer des personnes : un programme de carnet d'adresses pour les coordonnées de vos amis, un programme de gestion de contacts pour vos contacts professionnels ou un programme de suivi des employés. Bien que les programmes puissent être considérablement différents, les attributs qui s'appliquent à une personne restent les mêmes. Chaque personne a un nom, un âge, une adresse et un numéro de téléphone.

Dans cette leçon et les suivantes, vous allez créer une classe qui représente une personne ; vous pouvez enregistrer cette classe et l'utiliser dans les programmes que vous développerez ultérieurement.

Les classes peuvent être créées de trois manières : en tant que partie intégrante du code d'un module de formulaire dans un projet **Application Windows**, en tant que module de classe ajouté à un projet **Application Windows**, ou en tant que projet **Bibliothèque de classes** autonome.

Création de classes

Vous avez pu remarquer dans certaines leçons précédentes que lorsque vous double-cliquez sur un formulaire et ouvrez l'éditeur de code, un code semblable au suivant s'affiche :

```
Public Class Form1
    Private Sub Form1_Load...

    End Sub
End Class
```

Vous avez deviné : le formulaire est en fait une classe, marquée des instructions **Class** et **End Class**, et tout code entré entre ces deux instructions fait partie de la classe. Bien que par défaut un module de formulaire contienne une seule classe, vous pouvez créer des modules supplémentaires en ajoutant le code placé sous l'instruction **End Class** comme suit :

```
Public Class Form1
    ' Form1 code here
End Class
Public Class MyFirstClass
    ' Your class code here
End Class
```

L'inconvénient de ce procédé de création de classes tient dans ce que ces dernières sont alors uniquement disponibles dans le projet où vous les avez créées. Si vous souhaitez partager une classe avec d'autres projets, mieux vaut la placer dans un module de classe.

Modules de classe

Un module de classe est un fichier de code séparé qui contient une ou plusieurs classes. S'agissant d'un fichier séparé, il peut être réutilisé dans d'autres projets. Les modules de classe peuvent être créés de deux manières : en tant que module ajouté à un projet **Application Windows**, ou en tant que projet **Bibliothèque de classes** autonome.

Vous pouvez ajouter un nouveau module de classe à un projet existant en sélectionnant **Classe** dans la boîte de dialogue **Ajouter un nouvel élément**, accessible depuis le menu **Projet**. Dans le cadre du présent groupe de leçons, vous allez créer un projet **Bibliothèque de classes** autonome.

Essayez !

Pour créer un projet Bibliothèque de classes

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Bibliothèque de classes**.
3. Dans la zone **Nom**, tapez `Persons`, puis cliquez sur **OK**.

Un nouveau projet **Bibliothèque de classes** s'ouvre et l'éditeur de code affiche le module de classe `Class1.vb`.

4. Dans l'**Explorateur de solutions**, cliquez avec le bouton droit sur `Class1.vb` et sélectionnez **Renommer**, puis remplacez le nom par `Persons.vb`.

   Remarquez que le nom affiché dans l'éditeur de code est également devenu `Persons.vb`.

5. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

6. Dans la boîte de dialogue **Enregistrer un projet**, cliquez sur **Enregistrer**.

Conseil
Plutôt que d'enregistrer le projet dans l'emplacement par défaut, vous pouvez créer un répertoire dans lequel vous enregistrerez toutes vos classes à des fins de réutilisation. Vous pouvez saisir ce dossier dans le champ **Emplacement** de la boîte de dialogue **Enregistrer un projet** avant d'enregistrer.

   Pour le moment, laissez le projet ouvert ; vous lui ajouterez des éléments dans la leçon suivante.

Étapes suivantes

Dans cette leçon, vous avez appris comment créer un module de classe. Une classe vide a cependant peu d'intérêt ; dans la leçon suivante, vous apprendrez comment ajouter des propriétés à votre classe.

Leçon suivante : Ajout de propriétés à votre classe

Voir aussi

Tâches

Comment : ajouter de nouveaux éléments de projet

Concepts

Qu'est-ce qu'une classe ?

Autres ressources

Programmation avec des objets : utilisation de classes

Ajout de propriétés à votre classe

Dans cette leçon, vous apprendrez comment ajouter des propriétés à la classe que vous avez créée dans la leçon précédente.

Dans une leçon antérieure, Présentation détaillée : explication des propriétés, des méthodes et des événements, vous avez appris que tous les objets ont des attributs, et que les propriétés représentent des attributs. Dans la leçon précédente, vous avez créé une classe `Persons` qui représente une personne ; les gens ont des attributs tels que leur nom et leur âge, la classe `Persons` a donc besoin de propriétés pour représenter ces attributs.

Les propriétés peuvent être ajoutées à une classe de deux manières : en tant que *champ* ou en tant que *procédure de propriété*. Vous pouvez également déterminer comment une propriété fonctionne en utilisant ses modificateurs **Public**, **ReadOnly** et **WriteOnly**.

Champs et procédures de propriété

Les champs ne sont en réalité que des variables publiques au sein d'une classe qui peuvent être définies ou lues depuis l'extérieur de celle-ci. Les champs sont utiles aux propriétés qui ne doivent pas être validées ; par exemple, une valeur **Boolean** (**True** ou **False**). Dans le cas de la classe `Persons`, vous pouvez disposer d'une propriété **Boolean** nommée `Alive`, qui spécifie si une personne est vivante ou décédée. Sachant que seules deux valeurs sont possibles, un champ fonctionne parfaitement pour cette propriété.

Pour ajouter un champ à une classe, le code utilisé présenterait un aspect similaire à ce qui suit.

VB

```
Public Alive As Boolean
```

Toutefois, la plupart des propriétés sont plus complexes ; dans la plupart des cas, vous utiliserez une procédure de propriété pour ajouter une propriété à une classe. Les procédures de propriété présentent trois parties : une déclaration d'une variable privée pour stocker la valeur de propriété, une procédure **Get** qui expose la valeur, et une procédure **Set** qui définit la valeur.

Par exemple, une procédure de propriété destinée à une propriété `Name` pour la classe `Persons` présenterait l'aspect suivant :

VB

```
Private nameValue As String
Public Property Name() As String
    Get
        Name = nameValue
    End Get
    Set(ByVal value As String)
        nameValue = value
    End Set
End Property
```

La première ligne de code déclare une variable **String** privée, `nameValue`, qui enregistre la valeur de la propriété. La procédure de propriété proprement dite commence par `Public Property` et finit par `End Property`.

La procédure **Get** contient le code qui s'exécute lorsque vous souhaitez lire sa valeur ; par exemple, si vous lisez la propriété `Persons.Name`, le code renverra la valeur enregistrée dans la variable `nameValue`.

La procédure **Set** contient un code utilisé pour affecter une nouvelle valeur à la variable `nameValue`, à l'aide d'une valeur qui lui a été passée en tant qu'argument `value`. Par exemple, si vous écriviez le code `Persons.Name = "John"`, la valeur de type **String** `John` serait passée en tant qu'argument `value` ; le code présent dans la procédure **Set** l'affecterait alors à la variable `NameValue` pour l'enregistrer.

Vous pouvez vous demander pourquoi se compliquer la tâche alors que vous pouvez utiliser un champ pour représenter la propriété `Name`. Le monde réel impose certaines règles pour les noms ; par exemple, les noms ne contiennent généralement pas de nombres. Vous pourriez ajouter un code à la procédure **Set** pour vérifier l'argument `value` et pour retourner une erreur si celui-ci contenait des nombres.

Dans la procédure suivante, vous ajouterez un champ et trois propriétés à la classe `Persons`.

Essayez !

Pour ajouter des propriétés à votre classe

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Modélisation d'un objet réel : création de votre première classe, et réaliser les procédures qui s'y trouvent.
2. Dans l'**Explorateur de solutions**, sélectionnez **Persons.vb**, puis dans le menu **Affichage**, choisissez **Code**.
3. Ajoutez le code de déclaration suivant sous la ligne `Public Class Persons`.

   VB

   ```
   Private firstNameValue As String
   Private middleNameValue As String
   Private lastNameValue As String
   Public Alive As Boolean
   ```

4. Ajoutez les procédures de propriété suivantes sous le code de déclaration.

   VB

   ```
   Public Property FirstName() As String
       Get
           FirstName = firstNameValue
       End Get
       Set(ByVal value As String)
           firstNameValue = value
       End Set
   End Property

   Public Property MiddleName() As String
       Get
           MiddleName = middleNameValue
       End Get
       Set(ByVal value As String)
           middleNameValue = value
       End Set
   End Property

   Public Property LastName() As String
       Get
           LastName = lastNameValue
       End Get
       Set(ByVal value As String)
           lastNameValue = value
       End Set
   End Property
   ```

5. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Propriétés en lecture seule et en écriture seule

Parfois une propriété doit être définie une fois pour toutes et ne jamais être modifiée pendant l'exécution de votre programme. Par exemple, une propriété qui représente le numéro d'un d'employé ne doit jamais changer ; aussi pourra-t-elle être lue par un autre programme, sans que ce dernier puisse modifier sa valeur.

Le mot clé **ReadOnly** est utilisé pour spécifier qu'une valeur de propriété peut être lue, mais ne peut pas être modifiée. Si vous essayez d'affecter une valeur à une propriété **ReadOnly** , une erreur se produit dans l'éditeur de code.

Pour créer une propriété en lecture seule, vous créez une procédure de propriété avec une procédure **Get**, mais aucune procédure **Set**, comme illustré ci-dessous.

VB

```
Private IDValue As Integer
ReadOnly Property ID() As Integer
    Get
        ID = IDValue
    End Get
End Property
```

De la même manière, le mot clé **WriteOnly** autorise la définition d'une valeur de propriété, mais non sa lecture ; par exemple, vous n'autorisez pas la lecture d'une propriété de mot de passe par d'autres programmes. Vous pouvez manipuler cette valeur au sein de votre classe, mais vous souhaitez qu'elle reste privée.

Pour créer une propriété en écriture seule, vous créez une propriété avec une procédure **Set**, mais aucune procédure **Get**, comme illustré ci-dessous.

VB

```
Private passwordValue As String
WriteOnly Property Password() As String
    Set(ByVal value As String)
        passwordValue = value
    End Set
End Property
```

Les procédures de propriété **ReadOnly** et **WriteOnly** se révèlent également utiles lorsque vous souhaitez prendre une valeur de propriété et une seule, et la convertir en une valeur différente. Par exemple, l'âge d'une personne : contrairement à votre nom, votre âge évolue dans le temps. Donc si vous avez affecté votre âge à une classe et l'avez lu un an plus tard, la valeur lue est fausse.

Dans la classe `Persons`, vous pouvez empêcher cette situation en ajoutant deux propriétés ; une propriété **WriteOnly** `BirthYear` qui représente l'année de votre naissance, qui ne change jamais, et une propriété **ReadOnly** `Age` qui retourne une valeur en calculant la différence entre l'année actuelle et votre année de naissance.

Essayez !

Pour ajouter des propriétés ReadOnly et WriteOnly à votre classe

1. Ajoutez le code de déclaration suivant sous les autres déclarations présentes au début du module de classe.

   VB

   ```
   Private birthYearValue As Integer
   ```

2. Ajoutez les procédures de propriété suivantes sous le code de déclaration.

   VB

   ```
   WriteOnly Property BirthYear() As Integer
       Set(ByVal value As Integer)
           birthYearValue = value
       End Set
   End Property

   ReadOnly Property Age() As String
       Get
           Age = My.Computer.Clock.LocalTime.Year - birthYearValue
       End Get
   End Property
   ```

3. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris ce que sont les propriétés et les différentes manières de les ajouter à votre classe. Dans la leçon suivante, vous apprendrez comment ajouter des méthodes à votre classe afin que celle-ci puisse exécuter des actions.

Leçon suivante : Ajout de méthodes à votre classe

Voir aussi

Tâches

Modélisation d'un objet réel : création de votre première classe

Concepts

Procédures Property et champs

Autres ressources

Programmation avec des objets : utilisation de classes

Ajout de méthodes à votre classe

Dans cette leçon, vous apprendrez comment ajouter des méthodes à une classe afin qu'elle puisse effectuer des actions.

Dans une leçon précédente, Présentation détaillée : explication des propriétés, des méthodes et des événements, vous avez appris que la plupart des objets disposaient d'actions qu'ils peuvent effectuer ; ces actions sont connues sous le nom de méthodes. La classe `Persons` que vous avez créée dans la leçon Modélisation d'un objet réel : création de votre première classe représente une personne. Il est possible d'exécuter de nombreuses actions, et pour la classe `Persons`, ces actions peuvent être exprimées sous forme de méthodes de classe.

Méthodes de classe

Les méthodes d'une classe sont simplement des procédures **Sub** ou **Function** déclarées au sein de la classe. Par exemple, une classe `Account` peut disposer d'une procédure **Sub**, nommée `Recalculate`, qui mettrait à jour le solde d'un compte bancaire, ou d'une procédure `CurrentBalance` **Function** qui retournerait le solde le plus récent. Le code permettant de déclarer ces méthodes peut se présenter comme suit :

VB

```
Public Sub Recalculate()
    ' add code to recalculate the account.
End Sub
Public Function CurrentBalance(ByVal AccountNumber As Integer) As Double
    ' add code to return a balance.
End Function
```

Si la plupart des méthodes de classe sont publiques, vous pouvez également ajouter des méthodes utilisables uniquement par la classe elle-même. Par exemple, la classe `Persons` peut disposer de sa propre fonction pour calculer l'âge d'une personne. En déclarant la fonction comme étant de type **Private**, celle-ci ne peut être ni vue ni appelée depuis l'extérieur de la classe.

Le code d'une fonction privée peut présenter l'aspect suivant :

VB

```
Private Function CalcAge(ByVal year As Integer) As Integer
    CalcAge = My.Computer.Clock.LocalTime.Year - year
End Function
```

Si vous modifiez ultérieurement le code qui calcule la valeur de `CalcAge`, la méthode n'en fonctionnera pas moins parfaitement, et ce sans nécessiter aucune modification des codes qui l'utilisent. La dissimulation du code qui exécute la méthode est connue sous le nom d'*encapsulation*.

Dans la classe `Persons`, vous allez créer une méthode publique qui retourne un nom complet, et une fonction privée qui calcule l'âge.

Essayez !

Pour ajouter une méthode à votre classe

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Ajout de propriétés à votre classe, et réaliser les procédures qu'elle contient.
2. Dans l'**Explorateur de solutions**, sélectionnez **Persons.vb**, puis dans le menu **Affichage**, choisissez **Code**.
3. Ajoutez le code suivant sous les procédures de propriété.

   VB

   ```
   Public Function FullName() As String
       If middleNameValue <> "" Then
           FullName = firstNameValue & " " & middleNameValue & " " _
                                    & lastNameValue
       Else
   ```

```
            FullName = firstNameValue & " " & lastNameValue
        End If
    End Function
```

VB

```
    Private Function CalcAge(ByVal year As Integer) As Integer
        CalcAge = My.Computer.Clock.LocalTime.Year - year
    End Function
```

4. Modifiez le code dans la procédure de propriété `Age` de sorte à utiliser la fonction privée.

VB

```
    ReadOnly Property Age() As String
        Get
            ' Age = My.Computer.Clock.LocalTime.Year - birthDateValue
            Age = CalcAge(birthYearValue)
        End Get
    End Property
```

5. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris comment ajouter à la fois des méthodes publiques et privées à une classe. Vous pouvez approfondir vos connaissances sur les méthodes dans Présentation détaillée : création de plusieurs versions de la même méthode avec surcharge, ou passer à la leçon suivante et apprendre comment utiliser et tester la classe que vous avez créée.

Leçon suivante : Test de votre classe

Voir aussi

Tâches

Ajout de propriétés à votre classe

Autres ressources

Programmation avec des objets : utilisation de classes

Présentation détaillée : création de plusieurs versions de la même méthode avec surcharge

Dans cette leçon, vous apprendrez comment ajouter plusieurs versions d'une méthode à votre classe.

Dans la leçon précédente, vous avez appris comment ajouter des méthodes à la classe `Persons`. Dans certains cas, une méthode seule ne suffit pas. Par exemple, vous pouvez devoir passer différents types de données à la méthode selon des situations différentes, ou vous pouvez souhaiter retourner en conséquence des formats différents.

Vous pouvez créer plusieurs versions d'une méthode à l'aide d'une technique appelée *surcharge*. Lorsqu'une classe dispose de plusieurs méthodes dotées du même nom, mais de jeux d'arguments distincts, la méthode est dite surchargée.

Surcharge

Pour créer une méthode surchargée, ajoutez plusieurs procédures **Sub** ou **Function** à votre classe, portant chacune le même nom. Dans les déclarations de procédure, le jeu d'arguments de chaque procédure doit être distinct. Dans la négative, une erreur se produit.

Le code suivant illustre une méthode qui présente deux surcharges : une qui accepte un type **String** et l'autre qui accepte un type **Integer**, en tant qu'argument.

VB

```
Public Sub TestFunction(ByVal input As String)
    MsgBox(input)
End Sub
Public Sub TestFunction(ByVal input As Integer)
    MsgBox(CStr(input))
End Sub
```

Si vous deviez appeler cette méthode depuis votre code et lui passer une chaîne, la première surcharge serait exécutée et un message afficherait la chaîne ; si vous lui passiez un nombre, la seconde surcharge serait exécutée et le nombre serait converti en une chaîne, puis affiché dans le message.

Vous pouvez créer autant de surcharges que nécessaire ; chaque surcharge peut contenir un nombre différent d'arguments.

Dans la classe `Persons`, vous ajouterez une méthode avec deux surcharges pour retourner l'initiale du deuxième prénom d'une personne : une avec seulement l'initiale, l'autre avec l'initiale suivie d'un point.

Essayez !

Pour créer une méthode surchargée

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'avez pas enregistré, revenez à la leçon précédente, Ajout de méthodes à votre classe, et terminez les procédures.
2. Dans l'**Explorateur de solutions**, sélectionnez **Persons.vb**, puis dans le menu **Affichage**, choisissez **Code**.
3. Ajoutez le code suivant sous les méthodes existantes.

   VB

   ```
   Public Function MiddleInitial() As String
       MiddleInitial = Left$(middleNameValue, 1)
   End Function

   Public Function MiddleInitial(ByVal period As Boolean) As String
       MiddleInitial = Left$(middleNameValue, 1) & "."
   End Function
   ```

4. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris comment créer une méthode surchargée. Dans la leçon suivante, vous apprendrez comment utiliser la classe que vous avez créée dans un projet de test.

Leçon suivante : Test de votre classe

Voir aussi

Tâches

Ajout de méthodes à votre classe

Concepts

Considérations sur les surcharges de procédures

Autres ressources

Programmation avec des objets : utilisation de classes

Visite guidée de Visual Basic

Test de votre classe

Dans cette leçon, vous apprendrez comment créer une instance d'une classe pour tester cette dernière.

Dans les leçons précédentes, vous avez créé une classe `Persons` et lui avez attribué des propriétés et des méthodes. Jusqu'ici, vous vous êtes contenté d'ajouter du code. A présent, il est temps d'exploiter la classe `Persons` et de s'assurer qu'elle fonctionne correctement.

Création d'une Instance d'une Classe

Sans vous en rendre compte, vous avez utilisé des classes dans beaucoup de leçons précédentes. Les formulaires et les contrôles sont en réalité des classes ; lorsque vous faites glisser un contrôle **Button** jusqu'à un formulaire, vous créez en réalité une instance de la classe Button.

Toute classe peut être instanciée également dans votre code à l'aide d'une déclaration avec le mot clé **New**. Par exemple, pour créer une nouvelle instance de la classe **Button**, ajoutez le code suivant.

VB

```
Dim aButton As New Button
```

Pour utiliser et tester la classe `Persons`, vous devez d'abord créer un projet de test et ajouter une référence au module de classe.

Essayez !

Pour créer un projet de test pour votre classe

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Ajout de méthodes à votre classe, et réaliser les procédures qu'elle contient.
2. Dans le menu **Fichier**, pointez sur **Ajouter**, puis choisissez **Nouveau projet**.
3. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
4. Dans la zone **Nom**, tapez `PersonsTest`, puis cliquez sur **OK**.
5. Dans l'**Explorateur de solutions**, sélectionnez le projet `PersonsTest`, puis choisissez **Définir comme projet de démarrage** dans le menu **Projet**.
6. Dans l'**Explorateur de solutions**, sélectionnez le projet `PersonsTest`, puis dans le menu **Projet**, choisissez **Ajouter une référence**.

   La boîte de dialogue **Ajouter une référence** s'ouvre.
7. Cliquez sur l'onglet **Projets**, puis sélectionnez **Persons** et cliquez sur **OK**.
8. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code, puis entrez la déclaration suivante sous la ligne `Public Class Form1`.

   VB

   ```
   Dim person1 As New Persons.Persons
   ```

   Une nouvelle instance de la classe Persons est déclarée. Mais pourquoi avoir tapez deux fois Persons ? La première instance correspond au module de classe Persons.vb ; la seconde à la classe Persons au sein de ce module.
9. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Test de votre classe

L'étape suivante consiste à ajouter une interface utilisateur et un code qui utilise la classe `Persons`. Vous allez ajouter des zones de texte dans lesquelles l'utilisateur doit entrer des valeurs pour chacune des propriétés (sauf pour la propriété `Age` en lecture seule), une case à cocher pour le champ `Alive`, et des boutons pour tester chacune des méthodes publiques.

Essayez !

Pour tester votre classe

1. Dans l'**Explorateur de solutions**, sélectionnez Form1, puis dans le menu **Affichage**, choisissez **Concepteur**.
2. Depuis la **Boîte à outils**, faites glisser quatre contrôles **TextBox**, un contrôle **CheckBox** et deux contrôles **Button** dans le formulaire.
3. Sélectionnez le premier contrôle **Button**, puis dans la fenêtre **Propriétés**, affectez à sa propriété **Text** la valeur `Update`**.**
4. Sélectionnez le deuxième contrôle **Button**, puis dans la fenêtre **Propriétés**, affectez à sa propriété **Text** la valeur `Full Name`**.**
5. Double-cliquez sur le premier bouton (**Mise à jour**) pour ouvrir l'éditeur de code, puis dans le gestionnaire d'événements **Button1_Click**, ajoutez le code suivant :

   VB

   ```
   With person1
       .FirstName = Textbox1.Text
       .MiddleName = Textbox2.Text
       .LastName = Textbox3.Text
       .BirthYear = Textbox4.Text
       .Alive = CheckBox1.Checked
   End With
   ```

   Remarquez qu'à mesure de la saisie, une liste qui contient tous les membres de la classe `Persons` s'affiche. Sachant qu'elle a été ajoutée en tant que référence, IntelliSense affiche des informations sur votre classe comme s'il s'agissait de n'importe quelle autre classe.
6. Dans le gestionnaire d'événements **Button2_Click**, ajoutez le code suivant.

   VB

   ```
   ' Test the FullName method.
   MsgBox(person1.FullName)

   ' test the Age property and CalcAge method.
   MsgBox(CStr(person1.Age) & " years old")

   ' Test the Alive property.
   If person1.Alive = True Then
       MsgBox(person1.FirstName & " is alive")
   Else
       MsgBox(person1.FirstName & " is no longer with us")
   End If
   ```
7. Appuyez sur F5 pour exécuter le projet et afficher le formulaire.
   a. Dans la première zone de texte, tapez votre prénom.
   b. Dans la deuxième zone de texte, tapez votre deuxième prénom.
   c. Dans la troisième zone de texte, tapez votre nom.
   d. Dans la quatrième zone de texte, tapez les quatre chiffres de votre année de naissance (par exemple, 1983).
   e. Activez la case à cocher qui indique que vous êtes vivant.
8. Cliquez sur le bouton **Mise à jour** pour définir les propriétés de la classe, puis cliquez sur le bouton **Nom complet**.

   Trois messages s'affichent et présentent votre nom complet, votre âge et votre état.

9. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Test des méthodes surchargées

Si vous avez suivi la leçon facultative
Présentation détaillée : création de plusieurs versions de la même méthode avec surcharge, vous pouvez également tester les méthodes surchargées que vous avez ajoutées à la classe `Persons`. Si vous n'avez pas effectué cette leçon, revenez en arrière pour la suivre, ou ignorez la procédure suivante.

Essayez !

Pour tester les méthodes surchargées

1. Dans l'**Explorateur de solutions**, sélectionnez `Form1`, puis dans le menu **Affichage**, choisissez **Concepteur**.
2. À partir de la **Boîte à outils**, faites glisser deux autres contrôles **Button** dans le formulaire.
3. Sélectionnez le troisième contrôle **Button**, puis dans la fenêtre **Propriétés**, affectez à sa propriété **Text** la valeur `With`.
4. Sélectionnez le quatrième contrôle **Button**, puis dans la fenêtre **Propriétés**, affectez à sa propriété **Text** la valeur `Without`.
5. Double-cliquez sur le premier bouton (**With**) pour ouvrir l'éditeur de code, puis dans le gestionnaire d'événements **Button3_Click**, ajoutez le code suivant :

   VB
   ```
   MsgBox(person1.FirstName & " " & person1.MiddleInitial(True) & _
                            " " & person1.LastName)
   ```

   Remarquez qu'à mesure de la saisie, une liste qui contient tous les membres de la classe `Persons` s'affiche. Comme elle a été ajoutée comme référence, IntelliSense affiche des informations sur votre classe, comme il le ferait pour toute autre classe.
6. Dans le gestionnaire d'événements **Button4_Click**, ajoutez le code suivant.

   VB
   ```
   MsgBox(person1.FirstName & " " & person1.MiddleInitial & _
                            " " & person1.LastName)
   ```
7. Appuyez sur F5 pour exécuter le projet et afficher le formulaire.
   a. Dans la première zone de texte, tapez votre prénom.
   b. Dans la deuxième zone de texte, tapez votre deuxième prénom.
   c. Dans la troisième zone de texte, tapez votre nom.
   d. Dans la quatrième zone de texte, tapez les quatre chiffres de votre année de naissance (par exemple, 1983).
   e. Activez la case à cocher qui indique que vous êtes vivant.
8. Cliquez sur le bouton **Mise à jour** pour définir les propriétés de la classe, puis cliquez sur le bouton **With**.

   Un message affiche votre nom avec une virgule après l'initiale de votre second prénom.
9. Cliquez sur le bouton **Without**.

   Un message affiche votre nom avec une virgule après l'initiale de votre second prénom.
10. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Étapes suivantes

Dans cette leçon, vous avez appris comment créer un projet de test et comment l'utiliser pour tester les propriétés et les méthodes de votre classe. Dans la leçon suivante, vous apprendrez comment utiliser l'héritage pour créer une classe qui

repose sur une classe existante.

Leçon suivante : Génération de votre classe sur une classe existante : utilisation de l'héritage.

Voir aussi

Tâches

Ajout de méthodes à votre classe

Présentation détaillée : création de plusieurs versions de la même méthode avec surcharge

Génération de votre classe sur une classe existante : utilisation de l'héritage

Dans cette leçon, vous apprendrez comment utiliser l'héritage pour créer une classe qui repose sur une classe existante.

Nombre de projets réels ont en commun différents attributs et comportements. Par exemple, toutes les voitures ont des roues et un moteur, et peuvent rouler et s'arrêter (heureusement). Toutefois, certaines voitures ont des attributs qui sont propres à elles seules. Par exemple, un cabriolet a une partie supérieure amovible qui peut être repliée électroniquement ou à la main.

Si vous avez créé un objet pour représenter une voiture, vous avez intérêt à y intégrer des propriétés et des méthodes destinées à tous les attributs et comportements communs, et non à ajouter des attributs tels qu'un décapotage, sachant que cet attribut ne s'applique pas à toutes les voitures.

L'*héritage*, vous permet de créer une classe "cabriolet" dérivée de la classe voiture. Cette classe hérite de tous les attributs de la classe voiture, et permet d'ajouter les attributs et comportements uniques et propres à un cabriolet.

Héritage d'une classe existante

L'instruction **Inherits** permet de déclarer une nouvelle classe, appelée une *classe dérivée*, basée sur une classe existante, appelée une *classe de base*. Les classes dérivées héritent de tous les événements, propriétés, méthodes, champs et constantes définis dans la classe de base. Le code suivant illustre la déclaration pour une classe dérivée.

VB

```
Class DerivedClass
    Inherits BaseClass
End Class
```

La nouvelle classe, `DerivedClass`, peut alors être instanciée, ses propriétés et méthodes sont accessibles tout comme `BaseClass`, et vous pouvez ajouter de nouvelles propriétés et méthodes spécifiques à la nouvelle classe. Pour obtenir un exemple, recherchez à la classe `Persons` que vous avez créée dans les leçons précédentes.

Supposez que vous souhaitiez créer une classe qui représente des joueurs de base-ball ; ceux-ci disposent de tous des attributs définis dans la classe `Persons`, mais également d'attributs uniques, tels que leurs numéros et leurs positions. Plutôt que d'ajouter ces propriétés à la classe `Persons`, vous créez une nouvelle classe dérivée qui hérite de la classe `Persons`, puis ajoutez les nouvelles propriétés à cette classe.

Essayez !

Pour créer une classe dérivée

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, revenez à Test de votre classe, et effectuez les procédures.
2. Dans l'**Explorateur de solutions**, sélectionnez le nœud du projet **Persons**.
3. Dans le menu **Projet**, cliquez sur **Ajouter une classe**.
4. Dans la boîte de dialogue **Ajouter un nouvel élément**, tapez `Players` dans la zone **Nom**, puis cliquez sur **Ajouter**.

   Un nouveau module de classe est ajouté au projet.
5. Dans l'éditeur de code, ajoutez le code suivant juste sous la ligne `Public Class Players`.

   VB

   ```
   Inherits Persons
   ```
6. Ajoutez le code suivant pour définir deux nouvelles propriétés.

   VB

   ```
   Private numberValue As Integer
   ```

```
Private positionValue As String
Public Property Number() As Integer
    Get
        Number = numberValue
    End Get
    Set(ByVal value As Integer)
        numberValue = value
    End Set
End Property
Public Property Position() As String
    Get
        Position = positionValue
    End Get
    Set(ByVal value As String)
        positionValue = value
    End Set
End Property
```

7. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Test de la classe Players

Vous avez maintenant créé une classe `Players` dérivée de la classe `Persons`. Dans la procédure suivante, vous créerez un nouveau programme pour tester la classe `Players`.

Pour créer un projet de test pour votre classe

1. Dans le menu **Fichier**, pointez sur **Ajouter**, puis cliquez sur **Nouveau projet**.
2. Dans la boîte de dialogue **Ajouter un nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `PlayerTest`, puis cliquez sur **OK**.
4. Un nouveau projet Windows Forms est ajouté à l'**Explorateur de solutions**, et un nouveau formulaire s'affiche.
5. Dans l'**Explorateur de solutions**, sélectionnez le projet `PlayerTest`, puis choisissez **Définir comme projet de démarrage** dans le menu **Projet**.
6. Dans l'**Explorateur de solutions**, sélectionnez le projet `PlayerTest`, puis choisissez **Ajouter une référence** dans le menu **Projet**.

   La boîte de dialogue **Ajouter une référence** s'ouvre.
7. Cliquez sur l'onglet **Projets**, puis sélectionnez **Persons** et cliquez sur **OK**.
8. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code, puis entrez la déclaration suivante sous la ligne `Public Class Form1`.

   VB

```
Dim player1 As New Persons.Players
Dim player2 As New Persons.Players
```

9. Deux nouvelles instances de la classe `Players` sont déclarées.
10. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Pour tester la classe dérivée

1. Dans l'**Explorateur de solutions**, sélectionnez `Form1` dans le projet `PlayerTest`, puis dans le menu **Affichage**, choisissez **Code**.
2. Dans l'éditeur de code, ajoutez le code suivant à la procédure de l'événement **Form1_Load**.

VB

```
With player1
    .FirstName = "Andrew"
    .LastName = "Cencini"
    .Number = 43
    .Position = "Shortstop"
End With
With player2
    .FirstName = "Robert"
    .LastName = "Lyon"
    .Number = 11
    .Position = "Catcher"
End With
```

3. Dans l'**Explorateur de solutions**, sélectionnez `Form1` dans le projet `PlayerTest`, puis dans le menu **Affichage**, choisissez **Concepteur**.
4. À partir de la **Boîte à outils**, faites glisser deux contrôles **Button** jusqu'au formulaire.
5. Sélectionnez le premier contrôle **Button**, puis dans la fenêtre **Propriétés**, affectez à sa propriété **Text** la valeur `At Bat`.
6. Sélectionnez le deuxième contrôle **Button**, puis dans la fenêtre **Propriétés**, affectez à sa propriété **Text** la valeur `On Deck`.
7. Double-cliquez sur le premier bouton (`At Bat`) pour ouvrir l'éditeur de code, puis dans le gestionnaire d'événements **Button1_Click**, ajoutez le code suivant.

   VB

   ```
   MsgBox(player1.Position & " " & player1.FullName & ", #" & _
     CStr(player1.Number) & " is now at bat.")
   ```

   Remarquez que vous utilisez la méthode `FullName` qui a été héritée de la classe de base `Persons`.
8. Dans le gestionnaire d'événements **Button2_Click**, ajoutez le code suivant.

   VB

   ```
   MsgBox(player2.Position & " " & player2.FullName & ", #" & _
     CStr(player2.Number) & " is on deck.")
   ```
9. Appuyez sur F5 pour exécuter le programme. Cliquez sur chaque bouton pour examiner les résultats.
10. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Étapes suivantes

Dans cette leçon, vous avez appris comment hériter d'une classe existante et comment étendre la classe dérivée. Vous en savoir plus sur l'héritage, consultez Présentation détaillée : substitution de membres, ou passez à la leçon suivante pour apprendre ce que sont les collections.

Leçon suivante : Suivi d'éléments : utilisation de collections pour gérer plusieurs objets

Voir aussi

Concepts

Éléments fondamentaux de l'héritage

Présentation détaillée : substitution de membres

Dans cette leçon, vous apprendrez comment substituer un membre d'une classe dérivée.

Dans la leçon précédente, vous avez appris comment hériter d'une classe de base et étendre la classe dérivée avec de nouvelles propriétés. En plus d'ajouter de nouvelles propriétés ou méthodes à une classe dérivée, vous pouvez également souhaiter modifier, ou *substituer*, le comportement de propriétés ou de méthodes existantes.

Par exemple, vous pouvez créer une classe `Truck` dérivée d'une classe `Car` qui possède une méthode `StartEngine`. Si le camion (Truck) est équipé d'un moteur diesel, le processus du démarrage du moteur (StartEngine) peut différer de celui d'une voiture (Car) ; dans ce cas, vous souhaiterez substituer la méthode `StartEngine` pour mieux l'adapter au camion.

Substitution de propriétés et de méthodes

Par défaut, les propriétés et les méthodes d'une classe ne peuvent pas être substituées. Pour autoriser une classe dérivée à substituer une propriété ou une méthode, elle doit être marquée en tant que *substituable* par une déclaration au moyen du mot clé **Overridable**.

```
Public Overridable Property EngineType As String

Public Overridable Sub StartEngine(ByVal EngineType As String)
```

Lors de l'héritage d'une classe de base, les propriétés et les méthodes marquées comme étant **Overridable** peuvent être utilisées telles quelles, ou peuvent être modifiées pour répondre à vos besoins, par une déclaration à l'aide du mot clé **Overrides**.

```
Public Overrides Property EngineType As String

Public Overrides Sub StartEngine(ByVal EngineType As String)
```

Dans la classe `Players`, créée lors de la leçon précédente, vous pouvez substituer la méthode `FullName` pour inclure le numéro du joueur et éliminer le code qui retourne un deuxième prénom.

Essayez !

Pour substituer la méthode FullName

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'avez pas enregistré, revenez à la leçon précédente, Génération de votre classe sur une classe existante : utilisation de l'héritage, et terminez les procédures.
2. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Persons.vb**, puis dans le menu **Affichage**, sélectionnez **Code**.
3. Dans l'éditeur de code, modifiez la déclaration de la méthode `FullName` de la manière suivante :

   VB

   ```
   Public Overridable Function FullName() As String
   ```

4. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Players.vb**, puis dans le menu **Affichage**, sélectionnez **Code**.
5. Dans l'éditeur de code, ajoutez le code suivant à la classe.

   VB

   ```
   Public Overrides Function FullName() As String
       FullName = FirstName & " " & LastName & ", #" & numberValue
   End Function
   ```

6. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Form1.vb** dans le projet **PlayerTest**, puis dans le menu **Affichage**, sélectionnez **Code**.
7. Dans l'éditeur de code, modifiez le code de l'événement `Button1_Click` de la manière suivante :

   VB

   ```
   MsgBox(player1.Position & " " & player1.FullName & _
   ```

```
        " is now at bat.")
```

8. Modifiez le code de l'événement `Button2_Click` de la manière suivante :

   VB

```
MsgBox(player2.Position & " " & player2.FullName & _
       " is on deck.")
```

9. Appuyez sur F5 pour exécuter le programme et cliquez sur chaque bouton pour afficher les résultats.

   Remarquez que les résultats sont les mêmes qu'auparavant, bien que vous utilisiez désormais la méthode `FullName` substituée.

10. Dans le menu **Fichier**, sélectionnez **Enregistrer tout**.

Étapes suivantes

Dans cette leçon, vous avez appris comment substituer une méthode. Dans la leçon suivante, vous apprendrez comment utiliser des collections pour gérer des groupes d'objets similaires.

Leçon suivante : Suivi d'éléments : utilisation de collections pour gérer plusieurs objets

Voir aussi

Tâches

Génération de votre classe sur une classe existante : utilisation de l'héritage

Référence

Overridable

Overrides

Suivi d'éléments : utilisation de collections pour gérer plusieurs objets

Dans cette leçon, vous apprendrez comment utiliser une collection pour gérer des groupes d'objets.

Dans une leçon précédente, vous avez appris l'utilisation de tableaux pour gérer des groupes de variables. Même si vous pourriez utiliser des tableaux pour gérer les groupes d'objets, Visual Basic propose également un type d'objet spécial appelé *collection* qui peut être utilisé pour stocker et récupérer des groupes des mêmes objets.

Comme un tableau, chaque élément d'un objet Collection dispose d'un index qui permet de l'identifier. De plus, chaque élément d'un objet **Collection** dispose d'une *clé*, une valeur **String** qui permet de l'identifier. L'avantage que confère l'utilisation d'une clé tient dans ce que vous n'êtes pas contraint de mémoriser l'index d'un élément ; vous pouvez y faire référence au moyen d'un nom explicite.

Création d'une collection

Les collections sont utiles lorsque votre programme utilise plusieurs instances d'une même classe. Par exemple, examinez la classe `Players` que vous avez créée dans une leçon précédente. Vous aurez sans aucun doute besoin de plusieurs objets `Players` pour représenter une équipe de base-ball complète.

La première étape pour créer une collection consiste à créer une instance d'un objet **Collection**, comme illustré dans la déclaration suivante.

VB

```
Dim baseballTeam As New Collection
```

Une fois l'objet **Collection** créé, vous pouvez utiliser la méthode **Add** pour lui ajouter des éléments, et la méthode **Remove** pour supprimer ces éléments. Lors de l'ajout d'éléments, spécifiez en premier lieu l'élément à ajouter, puis spécifiez la valeur **String** à utiliser en tant que clé.

VB

```
baseballTeam.Add(playerObject, "Player's Name")
```

Lorsque vous supprimez un élément, utilisez la clé pour spécifier l'élément à supprimer.

VB

```
baseballTeam.Remove("Player's Name")
```

Dans la procédure suivante, vous ajouterez deux nouveaux objets `Players`. Ensuite, vous créerez une collection `team` et lui ajouterez les objets `Players` à l'aide de la propriété `Position` définie en tant que clé.

Essayez !

Pour créer une collection d'objets

1. Ouvrez le projet `Persons` que vous avez créé dans la leçon précédente. Si vous ne l'avez pas enregistré, revenez à la leçon précédente, Génération de votre classe sur une classe existante : utilisation de l'héritage, et terminez les procédures.
2. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Form1.vb** dans le projet **PlayerTest**, puis dans le menu **Affichage**, sélectionnez **Code**.
3. Dans l'éditeur de code, ajoutez le code suivant à la section des déclarations (sous la déclaration de `player2`).

   VB

   ```
   Dim player3 As New Persons.Players
   Dim player4 As New Persons.Players
   Dim team As New Collection
   ```

4. Ajoutez le code suivant à la procédure événementielle `Form1_Load`.

   VB

```
With player3
    .FirstName = "Eduardo"
    .LastName = "Saavedra"
    .Number = 52
    .Position = "First Base"
End With

With player4
    .FirstName = "Karl"
    .LastName = "Jablonski"
    .Number = 22
    .Position = "Pitcher"
End With

team.Add(player1, player1.Position)
team.Add(player2, player2.Position)
team.Add(player3, player3.Position)
team.Add(player4, player4.Position)
```

5. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Form1.vb** du projet **PlayerTest**. Ensuite, dans le menu **Affichage**, choisissez **Concepteur**.
6. À partir de la **Boîte à outils**, faites glisser un contrôle ComboBox dans le formulaire.
7. Dans la fenêtre **Propriétés**, sélectionnez la propriété **Éléments**, puis cliquez sur le bouton Sélection (**...**).
8. Dans l'**éditeur de collections String**, entrez le code suivant, puis cliquez sur **OK**.

```
Catcher

First Base

Pitcher

Shortstop
```

9. Double-cliquez sur le contrôle **ComboBox** pour ouvrir l'éditeur de code, puis dans le gestionnaire d'événements `ComboBox1_SelectedIndexChanged`, saisissez le code suivant.

```
Dim SelectedPlayer As Persons.Players
SelectedPlayer = team(ComboBox1.SelectedItem)
MsgBox("Playing " & ComboBox1.SelectedItem & " is " & _
SelectedPlayer.FullName & "!")
```

10. Appuyez sur F5 pour exécuter le programme. Sélectionnez un poste de jeu dans la liste déroulante ; le joueur à ce poste s'affiche dans un message.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser un objet **Collection** pour gérer un groupe d'objets. À ce stade, vous pouvez choisir d'en savoir plus sur les collections dans Présentation détaillée : utilisation d'une boucle For Each...Next dans une collection, ou de passer au groupe de leçons suivantes pour apprendre à créer vos propres contrôles.

Leçon suivante : Objets visibles : création de votre premier contrôle utilisateur

Voir aussi

Tâches

Présentation détaillée : utilisation d'une boucle For Each...Next dans une collection

Présentation détaillée : substitution de membres

Génération de votre classe sur une classe existante : utilisation de l'héritage

Concepts

Classe de la collection Visual Basic

Autres ressources

Objets visibles : création de votre premier contrôle utilisateur

Présentation détaillée : utilisation d'une boucle For Each...Next dans une collection

Dans cette leçon, vous apprendrez à utiliser une boucle **For Each...Next** pour appliquer une boucle à une collection.

Dans une leçon précédente, vous avez appris à utiliser une boucle **For...Next** pour exécuter un bloc de code un nombre de fois spécifié. Les objets de collection Visual Basic prennent en charge un type de boucle spécial, la boucle **For Each...Next**, que vous pouvez utiliser pour exécuter un bloc de code sur les différents éléments de la collection, plutôt qu'un nombre de fois défini.

Ajout d'une boucle For Each...Next

Dans la leçon précédente, vous avez ajouté manuellement les valeurs destinées à la propriété `Position` des objets `Players` de la collection de l'équipe au contrôle ComboBox. Si cette technique fonctionne dans le cadre de l'exemple, elle ne constitue pas la méthode conseillée ; à chaque fois que vous ajoutiez un nouveau joueur, vous deviez également mettre à jour la collection **Items** du contrôle **ComboBox**.

Une meilleure méthode pour gérer cette situation consiste à ajouter les valeurs `Position` à la collection **Items** en appliquant une boucle à la collection `team` à l'aide d'une boucle **For Each...Next**.

Dans une boucle **For...Next**, vous devez d'abord déclarer une variable de compteur ; avec une boucle **For Each...Next**, vous devez d'abord déclarer une variable objet. Le code ci-dessous illustre l'utilisation d'une boucle **For Each...Next**.

VB

```
Dim player As Persons.Players
For Each player In team
    ComboBox1.Items.Add(player.Position)
Next
```

Dans ce cas, peu importe combien de `Players` vous avez ; la méthode **ComboBox1.Items.Add** est exécutée une fois pour chaque objet `Players` dans la collection de l'équipe, et la valeur `Position` est ajoutée à la liste.

Essayez !

Pour appliquer une boucle à une collection

1. Ouvrez le projet `Persons` de la leçon précédente. Si vous ne l'avez pas terminé, revenez à la leçon précédente, Suivi d'éléments : utilisation de collections pour gérer plusieurs objets, et terminez les procédures.
2. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Form1.vb** dans le projet **PlayerTest**, puis dans le menu **Affichage**, sélectionnez **Concepteur**.
3. Sélectionnez le contrôle **ComboBox**. Ensuite, dans la fenêtre **Propriétés**, sélectionnez la propriété **Items** et cliquez sur le bouton Sélection (**...**).
4. Dans l'**éditeur de collections String**, supprimez les quatre entrées existantes, puis cliquez sur **OK**.
5. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
6. Dans l'éditeur de code, ajoutez le code suivant à la section des déclarations (sous la déclaration de `team`).

   VB

   ```
   Dim player As Persons.Players
   For Each player In team
       ComboBox1.Items.Add(player.Position)
   Next
   ```

7. Appuyez sur F5 pour exécuter le programme. Sélectionnez une position dans la liste déroulante. Le joueur qui occupe ce poste est affiché dans un message.

Étapes suivantes

Dans cette leçon, vous avez utilisé une boucle **For Each...Next** dans une collection. Dans les leçons suivantes, vous créerez un autre type d'objet ; un contrôle utilisateur.

Leçon suivante : Objets visibles : création de votre premier contrôle utilisateur

Voir aussi

Tâches

Suivi d'éléments : utilisation de collections pour gérer plusieurs objets

Référence

For Each...Next, instruction (Visual Basic)

Autres ressources

Programmation avec des objets : utilisation de classes

Objets visibles : création de votre premier contrôle utilisateur

Dans le jeu de leçons précédent, vous avez appris à utiliser des classes comme ébauches d'objets. Les classes que vous avez créées peuvent être réutilisées dans d'autres programmes afin que vous ne soyez plus obligé d'écrire le même code à plusieurs reprises.

Les contrôles sont également des classes qui peuvent être réutilisées dans plusieurs projets. Il est probable que vous deviez concevoir la même interface utilisateur à maintes reprises (par exemple en ajoutant des contrôles **TextBox** permettant d'entrer des prénoms et des noms, puis d'ajouter le code pour les combiner en un nom complet). Ne serait-ce pas agréable si vous pouviez éviter tout ce travail supplémentaire ?

C'est ici qu'interviennent les *contrôles utilisateur*. Vous pouvez considérer un contrôle utilisateur comme une classe permettant de créer des objets visibles (vos propres contrôles personnalisés qui peuvent être réutilisés comme les contrôles fournis avec Visual Basic 2005). La plupart des contrôles utilisateur sont des contrôles *composites*, c'est-à-dire des contrôles composés d'un ou plusieurs contrôles Visual Basic 2005 standard.

Dans les leçons suivantes, vous apprendrez comment créer un contrôle utilisateur composite que vous pouvez réutiliser dans d'autres programmes.

Dans cette section

Présentation du Concepteur de contrôles utilisateur

Ajout de contrôles à votre contrôle utilisateur

Ajout de code à votre contrôle utilisateur

Test de votre contrôle utilisateur

Rubriques connexes

Présentation détaillée : ajout des propriétés avec les valeurs nommées

Présentation détaillée : personnalisation de votre contrôle utilisateur

Présentation du Concepteur de contrôles utilisateur

Dans cette leçon, vous apprendrez comment créer votre propre contrôle à l'aide du **Concepteur de contrôles utilisateur**.

Dans le jeu de leçons précédent, vous avez appris comment utiliser un projet **Bibliothèque de classes** pour créer une classe. Un contrôle utilisateur est simplement une classe qui est visible. Comme les contrôles standard fournis avec Visual Basic 2005, un contrôle utilisateur peut être placé sur un formulaire au moment de la conception, et s'afficher lors de l'exécution du programme.

Lorsque vous concevez votre programme, vous utilisez le concepteur de formulaires pour organiser vos contrôles et déterminer leur aspect. Il existe également un concepteur pour les contrôles utilisateur (le **Concepteur de contrôles utilisateur**) qui vous permet, en tant que développeur, de déterminer l'aspect du contrôle.

Création de contrôles utilisateur

Un **Contrôle utilisateur** est semblable à toute autre classe, mais avec en outre la capacité d'être placé dans la **Boîte à outils** et affiché sur un formulaire. Alors qu'un module de **Classe** contient uniquement du code, un module de contrôle utilisateur possède à la fois du code et un concepteur. Le **Concepteur de contrôles utilisateur** est semblable à un concepteur de formulaires ; il possède des propriétés permettant de contrôler l'aspect et le comportement du contrôle utilisateur.

Les façons de créer un contrôle utilisateur diffèrent légèrement, selon la version de Visual Basic que vous utilisez : Visual Basic 2005 possède un type de projet **Bibliothèque de contrôles Windows** ; dans Visual Basic Express, vous devez d'abord créer un projet **Bibliothèque de classes**, puis ajouter un modèle **Contrôle utilisateur**.

Essayez !

Pour créer un contrôle utilisateur à l'aide de Visual Basic Express

1. Dans le menu **Fichier**, cliquez sur **Nouveau Projet**.
2. Dans le volet **Modèles** de la boîte de dialogue **Nouveau projet**, cliquez sur **Bibliothèque de classes**, puis cliquez sur **OK**.
3. Dans le menu **Projet**, choisissez **Ajouter un contrôle utilisateur**.
4. Dans la boîte de dialogue **Ajouter un nouvel élément**, sélectionnez **Contrôle utilisateur**.
5. Dans la zone **Nom**, tapez `NamesControl`, puis cliquez sur **Ajouter**.

   Un nouveau modèle **Contrôle utilisateur** est ajouté au projet et le **Concepteur de contrôles utilisateur** s'ouvre.
6. Dans l'**Explorateur de solutions**, cliquez avec le bouton droit sur `Class1.vb` et choisissez **Supprimer**, puis cliquez sur **OK**.
7. Dans le menu **Fichier**, cliquez sur **Enregistrer tout**.
8. Dans la boîte de dialogue **Enregistrer un projet**, tapez `NamesUserControl`, puis cliquez sur **Enregistrer**.

Pour créer un contrôle utilisateur à l'aide de Visual Studio 2005

1. Dans le menu **Fichier**, cliquez sur **Nouveau Projet**.
2. Dans le volet **Modèles** de la boîte de dialogue **Nouveau projet**, cliquez sur **Bibliothèque de contrôles Windows**.
3. Dans la zone **Nom**, tapez `NamesControl`, puis cliquez sur **OK**.

   Un nouveau modèle **Contrôle utilisateur** est ajouté au projet et le **Concepteur de contrôles utilisateur** s'ouvre.
4. Dans le menu **Fichier**, cliquez sur **Enregistrer tout**.
5. Dans la boîte de dialogue **Enregistrer un projet**, tapez `NamesUserControl`, puis cliquez sur **Enregistrer**.

Étapes suivantes

Dans cette leçon, vous avez appris comment créer un projet possédant un contrôle utilisateur et comment afficher le **Concepteur de contrôles utilisateur**. Cependant, un contrôle vierge n'est pas très utile. Dans la leçon suivante, vous apprendrez comment ajouter des contrôles à votre contrôle utilisateur pour créer son interface utilisateur.

Leçon suivante : Ajout de contrôles à votre contrôle utilisateur.

Voir aussi

Autres ressources

Objets visibles : création de votre premier contrôle utilisateur

Programmation avec des objets : utilisation de classes

Visite guidée de Visual Basic

Ajout de contrôles à votre contrôle utilisateur

Dans cette leçon, vous apprendrez comment ajouter des contrôles pour créer un contrôle utilisateur composite.

Conception de votre contrôle utilisateur

Comme mentionné précédemment, le type de contrôle utilisateur le plus courant est un contrôle composite (constitué d'un ou plusieurs contrôles Windows Forms standard). Vous pouvez ajouter des contrôles à un modèle **Contrôle utilisateur** en les faisant glisser depuis la **Boîte à outils** jusqu'au **Concepteur de contrôles utilisateur**, comme vous le feriez lors de la conception d'un formulaire.

Lorsque vous avez ajouté un contrôle, vous pouvez le redimensionner et le déplacer dans le concepteur, ainsi que définir ses propriétés dans la fenêtre **Propriétés**.

Pour cet exemple, vous ajouterez un contrôle **Label** pour afficher un nom complet, ainsi que trois contrôles **TextBox** pour entrer le prénom, le nom et le deuxième prénom.

Essayez !

Pour ajouter des contrôles à un contrôle utilisateur

1. Ouvrez le projet `NamesUserControl` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Présentation du Concepteur de contrôles utilisateur, et réaliser les procédures qui s'y trouvent.
2. Dans l'**Explorateur de solutions**, sélectionnez **NamesControl.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
3. À partir de la **Boîte à outils**, faites glisser un contrôle **Label** jusqu'au concepteur.

Conseil
La **Boîte à outils** est plus facile à utiliser si vous laissez la fenêtre ouverte. Pour ce faire, cliquez sur l'icône **Masquer automatiquement** qui ressemble à une punaise.

4. Dans la fenêtre **Propriétés**, affectez à la propriété **Name** la valeur `FullName`.
5. À partir de la **Boîte à outils**, faites glisser trois contrôles **TextBox** jusqu'au concepteur. Vous pouvez les organiser comme vous le souhaitez.
6. Dans la fenêtre **Propriétés**, remplacez la valeur de leur propriété **Name** par `FirstName`, `MiddleName` et `LastName`.
7. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris comment ajouter des contrôles dans le **Concepteur de contrôles utilisateur** et à définir leurs propriétés. Vous êtes à présent prêt à ajouter à votre contrôle afin qu'il exécute réellement une tâche utile. La leçon suivante montre comment écrire du code qui dirigera votre contrôle utilisateur pour afficher le prénom, le nom ainsi que le deuxième prénom, et qui exposera de nouvelles propriétés.

Leçon suivante : Ajout de code à votre contrôle utilisateur.

Voir aussi

Tâches

Présentation du Concepteur de contrôles utilisateur

Autres ressources

Objets visibles : création de votre premier contrôle utilisateur

Programmation avec des objets : utilisation de classes

Visite guidée de Visual Basic

Ajout de code à votre contrôle utilisateur

Dans cette leçon, vous apprendrez comment ajouter le code à votre contrôle utilisateur pour afficher le nom complet et exposer de nouvelles propriétés.

Comme les contrôles standard, les contrôles utilisateur possèdent des propriétés, des méthodes et des événements. En tant que développeur, vous écrirez le code permettant de gérer les événements de votre contrôle, et vous déterminerez les propriétés à exposer à l'utilisateur de votre contrôle.

Gestion des événements dans un contrôle utilisateur

Pour créer un contrôle utilisateur pratique, vous devrez écrire le code permettant de gérer les événements du contrôle. L'écriture d'une procédure de gestion des événements pour un contrôle utilisateur ne diffère pas de celle d'un formulaire ou d'un contrôle.

Dans cet exemple, vous écrirez une procédure événementielle qui mettra à jour l'étiquette `FullName` avec le contenu des zones `FirstName`, `MiddleName` et `LastName` que vous spécifiez, à l'aide du gestionnaire d'événements **TextChanged**.

Essayez !

Pour ajouter du code à un contrôle utilisateur

1. Ouvrez le projet `NamesUserControl` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Présentation du Concepteur de contrôles utilisateur, et réaliser les procédures qui s'y trouvent.
2. Dans l'**Explorateur de solutions**, sélectionnez **NamesControl.vb**, puis dans le menu **Affichage**, sélectionnez **Code**.
3. Dans l'éditeur de code, ajoutez le code suivant au gestionnaire d'événements **FirstName_TextChanged**.

```
Private Sub FirstName_TextChanged(ByVal sender As System.Object, ByVal e As System.Eve
ntArgs) Handles FirstName.TextChanged, MiddleName.TextChanged, LastName.TextChanged
    ' Display the contents of the three text boxes in the label.
    FullName.Text = FirstName.Text & " " & MiddleName.Text & " " & LastName.Text
End Sub
```

4. Appuyez sur F5 pour exécuter le programme. Le **Conteneur de test UserControl** s'ouvre et affiche votre contrôle utilisateur.
5. Entrez le prénom, le deuxième prénom et le nom dans les trois zones de texte. Au fur et à mesure que vous tapez, votre nom s'affiche dans l'étiquette `FullName`.

Si vous observez le code que vous avez entré ci-dessus, vous remarquerez que la clause **Handles** de la déclaration gère l'événement **TextChanged** pour les trois contrôles **TextBox**. Quelle que soit la zone de texte que vous remplissez en premier, l'étiquette `FullName` est toujours mise à jour au fur et à mesure que vous tapez.

Exposition de propriétés dans un contrôle utilisateur

Les propriétés des contrôles standard permettent de définir et de récupérer des valeurs pour un contrôle au moment de la conception et au moment de l'exécution. Vous pouvez également mettre certaines propriétés de votre contrôle utilisateur à disposition afin de pouvoir les définir dans la fenêtre **Propriétés** au moment de la conception et les référencer dans votre code.

L'exposition de propriétés dans un contrôle utilisateur est très similaire à l'exposition des propriétés dans une classe ; la principale différence réside dans le fait que vous pouvez également exposer les propriétés des contrôles contenus dans votre contrôle utilisateur. Comme avec les classes, vous devez déclarer une propriété et ajouter du code aux procédures **Get** et **Set**. Si vous exposez une propriété d'un contrôle contenu, vous n'avez pas besoin de déclarer une variable privée pour stocker la valeur ; la propriété du contrôle la stocke pour vous.

Il n'existe actuellement aucun moyen de récupérer le texte entré dans les contrôles `FirstName`, `MiddleName` et `LastName` ou la valeur de l'étiquette `FullName`. Vous devez exposer leur valeur comme propriétés pour rendre le contrôle plus utile. Comme vous ne souhaitez pas que la valeur de l'étiquette `FullName` soit modifiée en dehors de votre propre code, vous pouvez l'exposer en tant que propriété en lecture seule.

Essayez !

Pour ajouter des propriétés

1. Dans l'éditeur de code, ajoutez le code suivant pour exposer les valeurs `FirstName`, `MiddleName` et `LastName` en tant que propriétés.

```
Property FirstNameText() As String
    Get
        Return FirstName.Text
    End Get
    Set(ByVal value As String)
        FirstName.Text = value
    End Set
End Property
Property MiddleNameText() As String
    Get
        Return MiddleName.Text
    End Get
    Set(ByVal value As String)
        MiddleName.Text = value
    End Set
End Property
Property LastNameText() As String
    Get
        Return LastName.Text
    End Get
    Set(ByVal value As String)
        LastName.Text = value
    End Set
End Property
```

2. Ajoutez le code suivant pour exposer la valeur de l'étiquette `FullName` en tant que propriété en lecture seule.

```
ReadOnly Property FullNameText() As String
    Get
        Return FullName.Text
    End Get
End Property
```

3. Appuyez sur F5 pour exécuter le programme.
4. Dans le **Conteneur de test UserControl**, faites défiler jusqu'au bas de la grille **Propriétés** et sélectionnez la propriété **FirstNameText**. Entrez votre nom, puis sélectionnez propriété **FirstName** (la zone de texte **FirstName** doit afficher votre nom et la propriété **FullNameText** doit être identique).

   Essayez de modifier d'autres propriétés dans la grille **Propriétés** et dans le contrôle proprement dit pour observer la manière dont elles sont liées. C'est ce que pourra constater un utilisateur de votre contrôle au moment de la conception.
5. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris comment gérer des événements dans votre contrôle utilisateur et comment exposer certaines de ses propriétés. Vous pouvez approfondir vos connaissances sur les propriétés dans Présentation détaillée : ajout des propriétés avec les valeurs nommées, ou passer à la leçon suivante et apprendre comment utiliser le contrôle que vous avez créé.

Leçon suivante : Test de votre contrôle utilisateur.

Voir aussi
Tâches

Ajout de contrôles à votre contrôle utilisateur

Autres ressources

Objets visibles : création de votre premier contrôle utilisateur

Programmation avec des objets : utilisation de classes

Visite guidée de Visual Basic

Test de votre contrôle utilisateur

Dans cette leçon, vous apprendrez comment tester votre contrôle utilisateur dans un autre projet et observer son comportement au moment de l'exécution.

Comportement au moment de l'exécution

Lorsque vous avez terminé votre contrôle utilisateur et testé son comportement au moment du design dans le **Conteneur de test**, vous pouvez également observer son comportement en cas d'utilisation dans un programme. Visual Basic 2005 facilite le test de votre contrôle utilisateur en ajoutant un projet **Application Windows**.

Votre contrôle utilisateur s'affiche automatiquement dans la **Boîte à outils**, et vous pouvez l'ajouter à un formulaire ainsi que définir ses propriétés comme vous le feriez avec tout autre contrôle.

Essayez !

Pour tester votre contrôle utilisateur

1. Ouvrez le projet `NamesUserControl` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Ajout de code à votre contrôle utilisateur, et réaliser les procédures qui s'y trouvent.
2. Dans le menu **Fichier**, pointez sur **Ajouter**, puis cliquez sur **Nouveau projet**.
3. Dans la boîte de dialogue **Ajouter un nouveau projet**, sélectionnez **Application Windows**.
4. Dans la zone **Nom**, tapez `UserControlTest`, puis cliquez sur **OK**.

   Un nouveau projet est ajouté dans l'**Explorateur de solutions** et un nouveau formulaire s'affiche.
5. Dans l'**Explorateur de solutions**, sélectionnez le projet **UserControlTest**, et dans le menu **Projet**, sélectionnez **Définir comme projet de démarrage**.
6. Dans la **Boîte à outils**, sélectionnez **NamesControl** et faites-le glisser jusqu'au formulaire.
7. Dans la fenêtre **Propriétés**, définissez les propriétés **FirstNameText**, **MiddleNameText** et **LastNameText** afin qu'elles correspondent à votre nom.
8. Appuyez sur F5 pour exécuter le programme. Modifiez les noms contenus dans les zones de texte afin de vous assurer que l'étiquette se met correctement à jour.
9. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris comment ajouter votre contrôle utilisateur à un formulaire et observer son comportement au moment de l'exécution. À ce stade, vous pouvez choisir d'en savoir plus sur les contrôles utilisateur dans Présentation détaillée : personnalisation de votre contrôle utilisateur, ou de passer au groupe de leçons suivantes pour en savoir plus sur les graphiques.

Leçon suivante : Dessin d'images : utilisation de graphiques

Voir aussi

Tâches

Ajout de code à votre contrôle utilisateur

Autres ressources

Objets visibles : création de votre premier contrôle utilisateur
Programmation avec des objets : utilisation de classes
Visite guidée de Visual Basic

Présentation détaillée : personnalisation de votre contrôle utilisateur

Dans cette leçon, vous apprendrez comment personnaliser votre contrôle utilisateur pour le rendre plus pratique.

Ajout d'étiquettes

Dans la dernière leçon, vous avez testé le contrôle utilisateur **NamesControl** et vous avez constaté qu'il fonctionnait de la manière escomptée. Vous avez également pu noter de quelles manières il pourrait être amélioré. Par exemple, il n'est pas évident de déterminer le nom qui doit être entré dans chaque zone de texte, et il n'existe aucun moyen de vérifier que l'utilisateur a saisi les trois noms.

Pour que le contrôle utilisateur soit plus pratique, vous pouvez ajouter des étiquettes afin d'identifier chaque zone de texte. Vous pouvez définir le texte des étiquettes pour afficher « Prénom », « Deuxième prénom » et « Nom », mais que se passerait-il si vous décidiez ultérieurement d'utiliser plutôt l'étiquette « Initiale du deuxième prénom » ? Il est plutôt conseillé de créer des propriétés pour le texte des étiquettes afin de vous permettre de le modifier au moment de la conception et de spécifier une valeur par défaut pour chaque propriété.

Essayez !

Pour personnaliser votre contrôle utilisateur

1. Ouvrez le projet `NamesUserControl` que vous avez créé dans la leçon précédente. Si vous ne l'aviez pas enregistré, vous devez d'abord revenir à la leçon précédente, Test de votre contrôle utilisateur, et réaliser les procédures qui s'y trouvent.
2. Dans l'**Explorateur de solutions**, sélectionnez **NamesControl.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.
3. Dans la **Boîte à outils**, faites glisser trois contrôles **Label** jusqu'au concepteur et placez-en un au-dessus de chaque contrôle **TextBox**.
4. Dans l'**Explorateur de solutions**, sélectionnez **NamesControl.vb**, puis dans le menu **Affichage**, sélectionnez **Code**.
5. Dans l'éditeur de code, ajoutez le code suivant afin de créer des propriétés pour le texte des étiquettes.

```
Private text1 As String = "First Name"
Property Label1Text() As String
    Get
        Return text1
    End Get
    Set(ByVal value As String)
        text1 = value
        Label1.Text = text1
    End Set
End Property
Private text2 As String = "Middle Name"
Property Label2Text() As String
    Get
        Return text2
    End Get
    Set(ByVal value As String)
        text2 = value
        Label2.Text = text2
    End Set
End Property
Private text3 As String = "Last Name"
Property Label3Text() As String
    Get
        Return text3
    End Get
```

```
    Set(ByVal value As String)
        text3 = value
        Label3.Text = text3
    End Set
End Property
```

Remarquez que le code déclare trois variables **Private** pour le texte des étiquettes, et que les déclarations comprennent la valeur par défaut à afficher.

6. Dans l'éditeur de code, sélectionnez **(NamesControl Événements)** dans la zone déroulante de gauche, puis sélectionnez l'événement **Load** dans la zone déroulante de droite.

7. Ajoutez le code suivant au gestionnaire d'événements **NamesControl_Load** :

```
' Initialize the three labels
Me.Label1.Text = Label1Text
Me.Label2.Text = Label2Text
Me.Label3.Text = Label3Text
```

8. Dans le menu **Générer**, choisissez **Générer la solution**.

9. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.

   Vérifiez que les étiquettes affichent le texte par défaut. Essayez de modifier la propriété **Label1Text** dans la fenêtre **Propriétés** et vérifiez qu'elle change également dans le contrôle.

10. Dans le menu **Fichier**, choisissez **Fermer** pour fermer le concepteur de formulaires.

Ajout de la validation

Une autre personnalisation utile consisterait à ajouter du code permettant de *valider* les informations saisies afin de s'assurer qu'elles sont correctes. Plutôt que de valider chaque contrôle **TextBox** isolé, vous pouvez écrire le code de validation pour l'intégralité du contrôle utilisateur.

La plupart des contrôles possèdent un événement **Validating** qui est déclenché lorsque le focus quitte le contrôle ; c'est à cet endroit que vous devez insérer le code de validation. Dans ce cas-ci, vous souhaitez écrire le code permettant de vérifier que chaque zone de texte contient un nom.

Si une ou plusieurs zones de texte sont vides, vous pouvez afficher un message pour rappeler à l'utilisateur de saisir son nom. Vous pouvez exposer une propriété qui contient un message par défaut ; de cette manière, l'utilisateur de votre contrôle peut modifier le message afin qu'il indique les informations souhaitées.

Il est également possible que l'utilisateur de votre contrôle n'exige pas de deuxième prénom ; par conséquent, vous pouvez également ajouter une propriété **Boolean** permettant de désactiver la validation pour la zone de texte **MiddleName**.

Essayez !

Pour ajouter la validation

1. Dans l'éditeur de code, ajoutez du code pour deux propriétés liées à la validation : l'une pour spécifier si le deuxième prénom est requis et l'autre pour spécifier un message à afficher en cas d'échec de la validation.

```
Private required As Boolean = True
Property MiddleNameRequired() As Boolean
    Get
        Return required
    End Get
    Set(ByVal value As Boolean)
        required = value
    End Set
End Property
Private errormessage As String = "Please enter your name."
Property ValidationErrorMessage() As String
    Get
```

```
        Return errormessage
    End Get
    Set(ByVal value As String)
        errormessage = value
    End Set
End Property
```

2. Dans l'éditeur de code, sélectionnez **(NamesControl Événements)** dans la zone déroulante de gauche, puis sélectionnez l'événement **Validating** dans la zone déroulante de droite.

3. Ajoutez le code suivant au gestionnaire d'événements **NamesControl_Validating** :

```
If MiddleNameRequired = True Then
    If FirstName.Text = "" Or MiddleName.Text = "" Or _
LastName.Text = "" Then
        MsgBox(ValidationErrorMessage)
    End If
Else
    ' Middle name isn't required.
    If FirstName.Text = "" Or LastName.Text = "" Then
        MsgBox(ValidationErrorMessage)
    End If
End If
```

4. Dans le menu **Générer**, choisissez **Générer la solution**.

5. Dans l'**Explorateur de solutions**, sélectionnez **Form1.vb**, puis dans le menu **Affichage**, choisissez **Concepteur**.

   Sélectionnez le contrôle utilisateur sur le formulaire et vérifiez que les deux nouvelles propriétés s'affichent dans la fenêtre **Propriétés**.

6. À partir de la **Boîte à outils**, faites glisser un contrôle **Button** jusqu'au formulaire.

7. Appuyez sur F5 pour exécuter le programme.

   Entrez votre prénom et votre nom, mais ne saisissez pas votre deuxième prénom. Cliquez sur le bouton et un message contenant **ValidationErrorMessage** doit être affiché.

8. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer votre travail.

Étapes suivantes

Dans cette leçon, vous avez appris comment personnaliser votre contrôle utilisateur pour le rendre plus pratique. Dans la leçon suivante, vous apprendrez comment utiliser des graphiques pour dessiner des images et du texte.

Leçon suivante : Dessin d'images : utilisation de graphiques

Voir aussi

Tâches

Test de votre contrôle utilisateur

Autres ressources

Objets visibles : création de votre premier contrôle utilisateur
Programmation avec des objets : utilisation de classes
Visite guidée de Visual Basic

Dessin d'images : utilisation de graphiques

Dans une leçon précédente, vous avez appris comment créer l'interface utilisateur à l'aide de formulaires et de contrôles. Parfois, vous pouvez être amené à personnaliser l'aspect de votre programme au moyen d'images ou d'effets qui ne peuvent pas être créés avec des contrôles.

Visual Basic vous permet d'utiliser des *méthodes graphiques* pour dessiner pratiquement n'importe quel élément sur un formulaire ou un contrôle. Cet ensemble de leçons va vous présenter les capacités graphiques de Visual Basic.

Dans cette section

Rendre les graphiques apparents

Dessin de formes sur votre formulaire

Dessin de texte sur votre formulaire

Dessin d'une image sur votre formulaire

Rubriques connexes

Description : affichage d'images

Visite guidée de Visual Basic

Rendre les graphiques apparents

Dans cette leçon, vous apprendrez comment utiliser les méthodes graphiques dans Visual Basic pour dessiner sur un formulaire.

Dans une leçon précédente, vous avez appris comment afficher des images sur un formulaire à l'aide d'un contrôle **PictureBox**. Cette technique est efficace si vous disposez déjà d'une image, mais vous pouvez parfois être amené à dessiner directement sur votre formulaire. Par exemple, vous pouvez dessiner une ligne pour séparer deux champs, ou un cercle pour mettre en surbrillance une étiquette importante.

Visual Basic vous permet d'utiliser des *méthodes graphiques* pour dessiner pratiquement n'importe quel élément sur un formulaire ou un contrôle.

Notions graphiques de base

Avant de commencer à dessiner, vous devez savoir plusieurs choses. Un écran d'ordinateur est composé de milliers de très petits points, connus sous le nom de *pixels*. En définissant la couleur de chaque pixel, votre programme contrôle ce qui s'affiche à l'écran. Bien sûr, la plus grande part de cette tâche est réalisée pour vous par le code qui définit les formulaires et les contrôles.

Considérez le formulaire comme un canevas sur lequel vous pouvez dessiner ou peindre ; et comme un véritable canevas, un formulaire a des cotes. Là où un vrai canevas se mesure en centimètres, un formulaire se mesure en pixels. Un système de *coordonnées* détermine l'emplacement de chaque pixel ; la *coordonnée X* se mesure de gauche à droite et la *coordonnée Y* de haut en bas.

Les coordonnées démarrent dans l'angle supérieur gauche du formulaire. Aussi, si vous souhaitez dessiner un seul point à dix pixels à gauche et dix pixels au-dessous de ce point d'origine, vous spécifiez les coordonnées X et Y de la manière suivante : `10, 10`.

Les pixels sont également utilisés pour exprimer la largeur et la hauteur des éléments graphiques. Pour définir un carré de 100 pixels de large et de100 pixelsde haut, dont le coin supérieur gauche se trouve à 10 pixels du gauche et à 10 pixels du dessus, vous devez exprimer ses coordonnées comme suit : `10, 10, 100, 100`.

Le fait de dessiner à l'écran est connu sous le nom de *peinture*. Les formulaires et les contrôles disposent d'un événement **Paint** qui survient à chaque fois que ceux-ci doivent être redessinés ; par exemple, lorsqu'un formulaire s'affiche pour la première fois, ou lorsqu'il a été recouvert par une autre fenêtre. Tout code rédigé pour afficher des éléments graphiques est généralement contenu dans le gestionnaire d'événements **Paint**.

Dessin d'une ligne

Pour dessiner une ligne en travers d'un formulaire, vous devez définir deux éléments : ses coordonnées et sa couleur. Comme nous l'avons vu ci-dessus, les coordonnées X et Y sont exprimées en pixels. Une ligne donnée dispose de deux jeux de coordonnées : l'emplacement de début suivi de l'emplacement de fin.

Tout comme vous utilisez un stylo pour tracer une ligne sur un morceau de papier, Visual Basic utilise un objet **Pen** pour dessiner sur un formulaire. L'objet **Pen** définit l'aspect de la ligne ; dans le cas présent, sa couleur. Dans la procédure suivante, vous dessinerez des lignes horizontale, verticale et diagonale sur un formulaire.

Essayez !

Pour dessiner des lignes

1. Dans le menu **Fichier**, sélectionnez **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Lines`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code, puis sélectionnez **Paint** dans la liste déroulante **Événements**.
5. Dans le gestionnaire d'événements **Form1_Paint**, ajoutez le code suivant :

   VB

```
' Draw a 400 pixel black line 25 pixels from the top of the form.
e.Graphics.DrawLine(Pens.Black, 0, 25, 400, 25)
' Draw a 500 pixel red line 100 pixels from the left of the form.
e.Graphics.DrawLine(Pens.Red, 100, 0, 100, 500)
' Draw a diagonal blue line from the upper left to the lower right.
e.Graphics.DrawLine(Pens.Blue, 0, 0, Me.Width, Me.Height)
```

6. Appuyez sur F5 pour exécuter le programme. Le formulaire devrait afficher trois lignes.

Étapes suivantes

Dans cette leçon, vous avez appris les notions graphiques de base, ainsi que la manière de dessiner des lignes. Dans la leçon suivante, vous apprendrez à dessiner des formes telles que des rectangles et des cercles.

Leçon suivante : Dessin de formes sur votre formulaire

Voir aussi

Tâches

Description : affichage d'images

Autres ressources

Dessin d'images : utilisation de graphiques

Visite guidée de Visual Basic

Dessin de formes sur votre formulaire

Dans cette leçon, vous apprendrez comment dessiner des formes, telles que des rectangles ou des cercles, sur un formulaire.

Dans la leçon précédente, vous avez appris comment dessiner des lignes sur un formulaire à l'aide de la méthode graphique **DrawLine** et d'un objet **Pen**. En plus de la méthode **DrawLine**, Visual Basic dispose de méthodes graphiques pour dessiner des formes, et d'objets graphiques, connus sous le nom de *pinceaux* (brush en anglais), pour les remplir.

Dessin de formes simples

Le dessin d'une forme est une opération similaire au dessin d'une ligne ; vous devez définir les coordonnées et la couleur qui serviront au dessin. Là où une ligne obtient des coordonnées, qui définissent ses points de départ et de fin, une forme, telle qu'un carré ou un rectangle, obtient des coordonnées qui décrivent son coin supérieur gauche, sa largeur et sa hauteur.

Les cercles et les ovales (également connus sous le nom d'*ellipses*) n'ont pas de coin supérieur gauche ; aussi, les coordonnées décrivent les coins supérieurs gauches du rectangle les englobant : un rectangle imaginaire doté des mêmes largeur et hauteur que le cercle ou l'ovale.

Essayez !

Pour dessiner des formes

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, sélectionnez **Application Windows**.
3. Dans la zone **Nom**, tapez `Shapes`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code, puis sélectionnez **Paint** dans la liste déroulante **Événements**.
5. Dans le gestionnaire d'événements **Form1_Paint**, ajoutez le code suivant :

   VB

```
' Draw a 200 by 150 pixel green rectangle.
e.Graphics.DrawRectangle(Pens.Green, 10, 10, 200, 150)
' Draw a blue square
e.Graphics.DrawRectangle(Pens.Blue, 30, 30, 150, 150)
' Draw a 150 pixel diameter red circle.
e.Graphics.DrawEllipse(Pens.Red, 0, 0, 150, 150)
' Draw a 250 by 125 pixel yellow oval.
e.Graphics.DrawEllipse(Pens.Yellow, 20, 20, 250, 125)
```

6. Appuyez sur F5 pour exécuter le programme. Quatre formes doivent s'afficher dans le formulaire.

   Laissez le projet ouvert ; vous lui ajouterez des éléments dans la procédure suivante.

Dessin de formes pleines

Jusqu'à présent, les formes que vous dessiniez étaient de simples contours. Pour dessiner des formes avec les couleurs unies, vous devez utiliser l'une des méthodes *de remplissage*, notamment **FillRectangle** ou **FillEllipse**. Les méthodes de remplissage utilisent un objet **Brush**, un autre type d'objet graphique qui permet de peindre une surface.

Lorsque vous remplissez une forme d'une couleur différente, vous devez définir des coordonnées inférieures à celles de la forme, sinon sa bordure sera, elle aussi, couverte. Par exemple, pour remplir un carré de coordonnées `0, 0, 150, 150`, spécifiez un remplissage de coordonnées `1, 1, 148, 148`, en considérant une ligne de contour d'un pixel d'épaisseur.

Pour dessiner des formes pleines

1. Dans le gestionnaire d'événements **Form1_Paint**, ajoutez le code suivant sous le code que vous avez entré auparavant.

   VB

```
' Fill the circle with the same color as its border.
e.Graphics.FillEllipse(Brushes.Red, 0, 0, 150, 150)
' Fill the square with a different color.
e.Graphics.FillRectangle(Brushes.Aquamarine, 31, 31, 148, 148)
```

2. Appuyez sur F5 pour exécuter le programme.

   Notez que le carré rempli s'affiche sur le cercle rempli, mais que cette partie de sa bordure a disparu. L'ordre dans lequel vous appelez les méthodes graphiques détermine l'ordre dans lequel celles-ci dessinent ; dans le cas présent, le cercle rempli a été dessiné après le rectangle à bordure bleue.

   Modifiez l'ordre des méthodes et voyez le résultat.

Étapes suivantes

Dans cette leçon, vous avez appris comment dessiner et remplir des formes. Dans la leçon suivante, vous apprendrez comment dessiner un texte à l'aide de méthodes graphiques.

Leçon suivante : Dessin de texte sur votre formulaire

Voir aussi

Tâches

Rendre les graphiques apparents

Autres ressources

Dessin d'images : utilisation de graphiques

Visite guidée de Visual Basic

Dessin de texte sur votre formulaire

Dans cette leçon, vous apprendrez comment dessiner un texte sur un formulaire à l'aide de méthodes graphiques.

Dans une leçon précédente, vous avez appris comment afficher un texte à l'aide d'un contrôle **Label**. Toutefois, dans certains cas, vous pouvez être amené à dessiner vous-même un texte à l'aide de méthodes graphiques. Par exemple, si vous souhaitez un texte incliné, vous ne pouvez pas utiliser de contrôle **Label** ; en revanche, vous pouvez utiliser des méthodes graphiques pour dessiner le texte incliné selon l'angle de votre choix.

Dessin de texte

Pour dessiner un texte sur un formulaire ou un contrôle, utilisez la méthode graphique **DrawString**. Comme les autres méthodes de dessin, **DrawString** accepte un objet **Brush**. Ce dernier détermine la couleur du texte, ainsi que les coordonnées qui spécifient son emplacement ; dans le cas présent, les coordonnées X et Y du coin supérieur gauche du rectangle englobant le texte.

La méthode **DrawString** dispose également de deux arguments supplémentaires : la chaîne que vous souhaitez dessiner et la police qui détermine l'aspect du texte. Pour spécifier la police, vous devez en premier lieu créer un objet **Font**, puis utiliser cet objet en tant qu'argument pour la méthode **DrawString**.

Essayez !

Pour dessiner un texte

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.

   a. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.

   b. Dans la zone **Nom**, tapez `DrawText`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.

2. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code, puis sélectionnez **Paint** dans la liste déroulante **Événements**.

3. Dans le gestionnaire d'événements **Form1_Paint**, ajoutez le code suivant :

   VB

   ```
   ' Create a font object.
   Dim aFont As New System.Drawing.Font("Arial", 22, FontStyle.Bold)
   ' Display the text with the DrawString method.
   e.Graphics.DrawString("Graphics are fun!", aFont, Brushes.Black, _
       20, 10)
   ```

4. Appuyez sur F5 pour exécuter le programme. Le texte devrait s'afficher dans le formulaire.

   Laissez le projet ouvert ; vous lui ajouterez des éléments dans la procédure suivante.

Dessin d'un texte qui a fait l'objet d'une rotation

Pour dessiner un texte selon un angle donné, vous devez utiliser un autre type de méthode graphique appelé *transformation*. Il existe plusieurs types de transformations disponibles pour différents effets graphiques ; dans le cas présent, vous utiliserez la méthode **RotateTransform**.

La méthode **RotateTransform** prend un argument unique : l'angle de rotation du texte. La transformation s'effectue sur la ligne de code qui suit la méthode **RotateTransform** ; vous pouvez également l'utiliser pour faire pivoter des formes ou des lignes dessinées à l'aide d'autres méthodes de dessin.

Essayez !

Pour dessiner un texte ayant fait l'objet d'une rotation

1. Dans le gestionnaire d'événements **Form1_Paint**, ajoutez le code suivant sous le code que vous avez entré auparavant.

VB

```
' Rotate the text 45 degrees.
e.Graphics.RotateTransform(45)
e.Graphics.DrawString("And exciting too!", aFont, Brushes.Red, _
    100, 0)
```

2. Appuyez sur F5 pour exécuter le programme. Le texte tourné devrait s'afficher dans le formulaire.

Étapes suivantes

Dans cette leçon, vous avez appris comment utiliser la méthode **DrawString** pour afficher un texte. Dans la leçon suivante, vous apprendrez à afficher une image à l'aide de méthodes graphiques.

Leçon suivante : Dessin d'une image sur votre formulaire

Voir aussi

Tâches

Dessin de formes sur votre formulaire

Autres ressources

Dessin d'images : utilisation de graphiques

Visite guidée de Visual Basic

Dessin d'une image sur votre formulaire

Dans cette leçon, vous apprendrez comment afficher une image à l'aide d'appels graphiques.

Dans une leçon précédente, vous avez appris comment afficher une image à l'aide d'un contrôle **PictureBox**. Il est également possible d'afficher une image à partir d'un fichier à l'aide des méthodes graphiques de Visual Basic. Comme dans la leçon précédente, l'utilisation de méthodes graphiques plutôt que d'un contrôle **PictureBox** est une nécessité si vous souhaitez effectuer des opérations spéciales, telles que l'application d'une rotation à l'image.

Affichage d'une image

Pour afficher une image sur un formulaire ou un contrôle, utilisez la méthode graphique **DrawImage**. La méthode **DrawImage** accepte une image bitmap en tant qu'argument, ainsi que les coordonnées X et Y qui définissent l'angle supérieur gauche de l'image.

Essayez !

Pour afficher une image qui a fait l'objet d'une rotation

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `DrawImage` , puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Dans l'**Explorateur de solutions**, double-cliquez sur le nœud **Mon projet** pour ouvrir le **Concepteur de projets**.
5. Dans le **Concepteur de projets**, cliquez sur l'onglet **Ressources**, sélectionnez **Ajouter une ressource**, puis sélectionnez **Ajouter un fichier existant**.
6. Dans la boîte de dialogue **Ajouter le fichier existant aux ressources**, naviguez jusqu'à un fichier image et sélectionnez-le, puis cliquez sur **Ouvrir**.
7. Dans l'**Explorateur de solutions**, sélectionnez le nœud **Form1**, puis dans le menu **Affichage**, sélectionnez **Code** pour ouvrir l'éditeur de code.
8. Dans l'éditeur de code, sélectionnez **Paint** dans la liste déroulante **Événements**.
9. Dans le gestionnaire d'événements **Form1_Paint**, ajoutez le code suivant :

   VB

```
e.Graphics.RotateTransform(45)
e.Graphics.DrawImage(My.Resources.picture, 50, 0)
```

   Remarque

   Remplacez `picture` par le nom de la ressource que vous avez ajoutée dans une étape antérieure.
10. Appuyez sur F5 pour exécuter le programme. L'image qui a fait l'objet d'une rotation devrait s'afficher dans le formulaire.

Étapes suivantes

Dans cette leçon, vous avez appris comment afficher dans un formulaire une image qui a fait l'objet d'une rotation. Cette leçon termine la série sur les graphiques ; dans la série de leçons suivante, vous apprendrez comment partager les programmes que vous créez.

Leçon suivante : Distribution de votre programme

Voir aussi

Tâches

Dessin de texte sur votre formulaire

Autres ressources

Dessin d'images : utilisation de graphiques
Visite guidée de Visual Basic

Distribution de votre programme

Il y a plusieurs manières de partager votre programme avec les autres utilisateurs. En utilisant la publication ClickOnce, vous pouvez mettre votre programme à disposition sur un CD-ROM ou un DVD-ROM ; vous pouvez également le publier sur un site Web où d'autres utilisateurs pourront le télécharger et l'exécuter. Vous pouvez également l'envoyer par courrier électronique, ou simplement le copier sur un disque.

Dans cette section

Partage de votre programme : introduction au déploiement

Distribution de votre programme sur CD : publication avec ClickOnce

Rubriques connexes

Présentation détaillée : ajout de composants requis dans votre programme

Présentation détaillée : distribution de votre programme sur Internet

Envoi de votre programme par messagerie électronique : création d'un fichier compressé

Obtention d'un programme par messagerie électronique : extraction d'un programme compressé

Partage de votre programme : introduction au déploiement

Dans cette leçon, vous apprendrez les différentes manières de partager vos programmes avec d'autres utilisateurs.

Une fois votre programme écrit, testé et débogué, vous souhaiterez probablement le partager avec d'autres utilisateurs. Le processus de copie et de distribution de votre programme est connu sous le nom de *déploiement*.

Si vous pensez qu'il s'agit simplement de copier les fichiers du programme et d'exécuter ce dernier sur un autre ordinateur, vous constaterez qu'avec cette approche, dans nombre de cas, le programme ne s'exécute pas. En effet, la plupart des programmes dépendent d'autres éléments logiciels, connus sous le nom de *composants*, qui doivent être installés sur l'ordinateur de destination. Si vous tentez d'exécuter votre programme alors qu'un composant est absent, votre programme ne s'exécute pas.

Publication ClickOnce

Visual Basic intègre des outils qui permettent de déployer votre programme à l'aide d'une méthode connue sous le nom de publication ClickOnce. Celle-ci facilite le déploiement de votre programme, en incluant et en installant automatiquement tous les composants qui lui sont nécessaires. ClickOnce vous permet de publier votre programme sur un CD-ROM ou un DVD-ROM, que vous pouvez alors partager avec d'autres utilisateurs.

Si vous avez accès à un serveur Web, vous pouvez également utiliser ClickOnce pour publier votre programme sur un site Web ; le programme peut alors être téléchargé sur Internet. Si vous modifiez votre programme, vous pouvez publier sa nouvelle version. Tout utilisateur ayant téléchargé votre programme peut alors télécharger automatiquement cette nouvelle version à la prochaine exécution du programme.

Pour publier un programme à l'aide de ClickOnce, vous devez disposer d'un accès à un graveur de CD-ROM ou de DVD-ROM, ou à un serveur Web. Si vous n'avez accès à aucun de ces supports, vous pouvez toujours partager votre programme en le compressant, puis en le copiant sur une disquette ou en l'envoyant par courrier électronique.

Étapes suivantes

Dans cette rubrique, vous avez appris le déploiement et la publication ClickOnce. Dans la leçon suivante, vous apprendrez comment publier votre programme à l'aide de ClickOnce.

Leçon suivante : Distribution de votre programme sur CD : publication avec ClickOnce

Voir aussi

Tâches

Envoi de votre programme par messagerie électronique : création d'un fichier compressé

Autres ressources

Distribution de votre programme

Distribution de votre programme sur CD : publication avec ClickOnce

Dans cette leçon, vous apprendrez comment publier votre programme sur un CD-ROM ou un DVD pour le partager avec d'autres utilisateurs.

Pour partager votre programme avec les autres, vous devez d'abord créer un programme d'installation et le copier sur un CD-ROM ou un autre support. En plus du programme proprement dit, le programme d'installation doit inclure d'autres composants ou fichiers nécessaires à l'exécution de votre application ; ces éléments sont connus sous le nom de *composants requis*.

Comme vous pouvez l'imaginer, déterminer avec précision les composants requis n'est pas nécessairement une tâche facile ; heureusement les outils de publication ClickOnce de Visual Basic effectuent cette opération et même d'autres pour vous. Ainsi, ClickOnce permet également de publier votre programme sur Internet ou sur un réseau local. Toutefois, dans cette leçon, vous allez publier votre programme sur un CD-ROM.

Remarque

Pour publier un programme sur un CD-ROM ou un DVD, l'ordinateur utilisé pour le développer doit disposer du matériel et des logiciels appropriés. Si vous pouvez graver des CD-ROM de musique, vous devez être en mesure de publier un program me sur un CD-ROM. Si vous ne disposez pas de cette capacité, vous pouvez cependant être en mesure de partager votre pro gramme par un autre biais ; pour plus d'informations, consultez
Envoi de votre programme par messagerie électronique : création d'un fichier compressé.

Publication avec ClickOnce

La publication de votre programme à l'aide de ClickOnce est un processus assez simple. Il implique quelques choix dans l'**Assistant Publication**, puis la gravure des fichiers résultants sur un CD-ROM.

Avant de publier votre programme, assurez-vous que vous l'avez testé et qu'il s'exécute sans erreur. Une fois que vous êtes prêt à le publier, vous pouvez lancer l'**Assistant Publication** en sélectionnant la commande **Publier** dans le menu **Générer**.

L'**Assistant Publication** comprend trois étapes : La première étape consiste à sélectionner l'emplacement où vous souhaitez placer le programme d'installation et tous les fichiers associés. Si vous publiez votre programme sur un CD-ROM, sélectionnez un dossier sur votre disque local ; vous sélectionnerez à nouveau cet emplacement pour graver le programme sur un CD-ROM. La deuxième étape consiste à spécifier la manière dont les utilisateurs installeront votre programme ; dans le cas présent, à partir d'un CD-ROM.

L'étape finale implique le fait de spécifier si votre programme vérifie ou non automatiquement à chaque démarrage la présence d'une version plus récente. Si vous avez accès à un serveur Web, vous pouvez publier des versions mises à jour de votre programme, comme le décrit Présentation détaillée : distribution de votre programme sur Internet. Toutefois, dans le cas présent, sachant que vous effectuez votre publication depuis un CD-ROM, votre programme n'a aucun moyen de vérifier la présence de mises à jour.

Lorsque l'**Assistant Publication** s'exécute, il détermine automatiquement tout composant requis éventuel nécessaire à votre programme. Par défaut, des composants requis, tels que le .NET Framework, ne sont pas réellement intégrées au programme d'installation ; au cours de l'installation, ce dernier s'assure de la présence des composants requis. Si nécessaire, il les télécharge depuis Internet, puis les installe.

Pour intégrer des composants requis au programme d'installation enregistré sur le CD-ROM, vous pouvez définir une propriété dans le **Concepteur de projets**. Cependant, vous devez d'abord télécharger les fichiers redistribuables de ces composants requis sur votre ordinateur local. Pour en savoir plus, consultez
Présentation détaillée : ajout de composants requis dans votre programme.

Essayez !

Pour publier sur un CD-ROM

1. Dans le menu **Fichier**, sélectionnez **Ouvrir un projet**.
2. Dans la boîte de dialogue **Ouvrir un projet**, naviguez jusqu'à un projet **Application Windows** créé au cours d'une leçon précédente, puis cliquez sur **Ouvrir**.
3. Appuyez sur F5 pour exécuter le projet. Vous devez résoudre les erreurs éventuelles avant de poursuivre.

4. Dans le menu **Déboguer**, choisissez **Arrêter le débogage**.
5. Dans le menu **Générer**, sélectionnez **Publier NomProjet**, où **NomProjet** est le nom de votre projet.

   L'**Assistant Publication** démarre.
6. Dans la page **Où souhaitez-vous publier l'application ?** de l'**Assistant Publication**, entrez le chemin d'accès vers l'endroit où vous souhaitez publier votre programme. Par exemple, `C:\My Programs`. Si le dossier n'existe pas, vous êtes invité à le créer.

   Cliquez sur **Suivant** pour passer à la page suivante de l'Assistant.
7. Dans la page **Comment les utilisateurs installeront-ils l'application ?**, sélectionnez **À partir d'un CD-ROM ou DVD-ROM**, puis cliquez sur **Suivant**.
8. Dans la page **Où l'application doit-elle vérifier la disponibilité de mises à jour ?**, sélectionnez **L'application ne vérifiera pas la disponibilité de mises à jour**.
9. Cliquez sur **Terminer**. Le programme sera publié à l'endroit spécifié dans la première page de l'Assistant.

   Vous pouvez à présent utiliser votre application de gravure de CD-ROM ou de DVD-ROM pour graver un CD-ROM ou un DVD-ROM pour votre programme. Vous devez inclure tous les fichiers dans le dossier où vous avez publié le programme.
10. Installez le CD-ROM ou le DVD-ROM fini sur un autre ordinateur, puis exécutez le fichier Setup.exe. Si certains composants requis, tels que le .NET Framework, doivent être installés, vous êtes invité à les télécharger et à les installer.
11. Une fois l'installation terminée, vous pouvez exécuter le programme à partir du raccourci placé dans le menu **Démarrer**.

    Pour savoir comment intégrer des composants requis à votre programme, ne fermez pas le projet ; vous allez l'utiliser dans la leçon Présentation détaillée : ajout de composants requis dans votre programme.

Étapes suivantes

Dans cette leçon, vous avez appris comment publier un programme sur un CD-ROM à l'aide de la publication ClickOnce. À ce stade, plusieurs suites sont possibles.

Si vous disposez d'un accès à un serveur Web et si vous souhaitez apprendre comment publier votre programme sur Internet, consultez Présentation détaillée : distribution de votre programme sur Internet.

Si vous n'étiez pas en mesure de graver un CD-ROM ou un DVD-ROM, consultez
Envoi de votre programme par messagerie électronique : création d'un fichier compressé.

Si vous souhaitez apprendre comment intégrer des composants requis à votre programme, consultez
Présentation détaillée : ajout de composants requis dans votre programme.

Sinon, vous avez terminé la partie principale de la visite guidée. Vous pouvez passer à la rubrique suivante pour consulter quelques suggestions quant à des ressources pédagogiques relatives à Visual Basic.

Leçon suivante : Étape suivante : que faire ensuite ?

Voir aussi
Concepts
Partage de votre programme : introduction au déploiement

Présentation détaillée : ajout de composants requis dans votre programme

Dans cette leçon, vous apprendrez à créer un package qui contient votre programme ainsi que les différents composants qu'il requiert, à l'aide de la publication ClickOnce.

Par défaut, les programmes publiés au moyen de la technologie ClickOnce téléchargent tous les composants requis sur Internet pendant installation. Si vous tentez d'installer le programme sur un ordinateur dépourvu d'accès à Internet, l'installation peut échouer.

Pour empêcher une telle situation, vous pouvez créer un package qui contiendra à la fois les composants requis, tels que les fichiers redistribuables du .NET Framework, et le programme.

Remarque
Si vous savez que tous les utilisateurs qui installent votre programme disposent d'un accès à Internet, vous pouvez utiliser la méthode par défaut ; ainsi, si une version plus récente d'un composant requis est disponible, les utilisateurs en bénéficient.

Intégration de composants requis

Pour intégrer des composants requis à votre programme, modifiez la propriété **Publishing** dans le **Concepteur de projets**. Remarquez que si vous incluez un composant requis, vous devez les inclure tous ; vous ne pouvez pas inclure un seul composant requis et obtenir les autres par téléchargement sur Internet.

Essayez !

Pour intégrer des composants requis

1. Ouvrez votre projet de la leçon précédente, Distribution de votre programme sur CD : publication avec ClickOnce.
2. Dans l'**Explorateur de solutions**, sélectionnez le nœud du projet, et dans le menu **Projet**, choisissez **Propriétés**.

   Le **concepteur de projets** s'ouvre.
3. Dans le **Concepteur de projets**, cliquez sur l'onglet **Publier**.
4. Cliquez sur le bouton **Composants requis** pour ouvrir la boîte de dialogue **Composants requis**.
5. Dans la boîte de dialogue **Composants requis**, activez la case à cocher **Télécharger les composants requis à partir de l'emplacement de mon application**, puis cliquez sur **OK**
6. Dans le menu **Générer**, sélectionnez **Publier NomProjet**, où **NomProjet** est le nom de votre projet.

   L'**Assistant Publication** s'exécute.
7. Cliquez sur **Terminer** pour publier le programme.

Remarque
Vous pouvez être invité à télécharger les fichiers redistribuables du .NET Framework si vous ne l'avez pas encore fait.

8. Dans l'Explorateur Windows, accédez à l'emplacement où le programme a été publié, et vérifiez que les fichiers Setup.exe et Dotnetfx.exe sont présents.

Étapes suivantes

Dans cette leçon, vous avez appris à publier des composants requis avec votre programme. À ce stade, plusieurs suites sont possibles.

Si vous disposez d'un accès à un serveur Web et si vous souhaitez apprendre comment publier votre programme sur Internet, consultez Présentation détaillée : distribution de votre programme sur Internet.

Si vous n'étiez pas en mesure de graver un CD-ROM ou un DVD-ROM, consultez
Envoi de votre programme par messagerie électronique : création d'un fichier compressé.

Sinon, vous avez terminé les leçons de la visite guidée. Vous pouvez passer à la rubrique suivante pour consulter quelques

suggestions quant à d'autres ressources pédagogiques relatives à Visual Basic.

Leçon suivante : Étape suivante : que faire ensuite ?

Voir aussi

Tâches

Distribution de votre programme sur CD : publication avec ClickOnce

Concepts

Partage de votre programme : introduction au déploiement

Présentation détaillée : distribution de votre programme sur Internet

Dans cette leçon, vous apprendrez comment utiliser la publication ClickOnce pour déployer votre programme sur un serveur Web.

Dans la leçon précédente, vous avez appris à publier un programme sur un CD-ROM ou un DVD-ROM. Si vous avez accès à un serveur Web, vous pouvez également utiliser ClickOnce pour publier votre programme afin de le rendre disponible sur Internet.

En plus de rendre le programme disponible à grande échelle, la publication sur un serveur Web vous permet de tirer parti des fonctions de mise à jour automatique de ClickOnce. Si vous publiez ultérieurement une nouvelle version de votre programme, lorsqu'un utilisateur tente à nouveau de l'exécuter, le programme l'invite à télécharger et à installer la nouvelle version.

Remarque
Pour publier votre programme sur un serveur Web, ce dernier doit exécuter IIS (Internet Information Services), les extensions FrontPage doivent être installées, et vous devez disposer de privilèges d'administration dans IIS.

Publication sur un serveur Web

La publication sur un serveur Web est très semblable à la publication sur un CD-ROM ou un DVD-ROM ; vous devez simplement faire quelques choix supplémentaires dans l'**Assistant Publication**.

L'un d'eux implique de spécifier si le programme sera disponible hors connexion ; c'est-à-dire lorsque l'ordinateur n'est pas connecté à Internet. Si vous rendez le programme disponible à la fois en ligne et hors connexion, une entrée sera ajoutée au menu **Démarrer** de Windows afin que l'utilisateur puisse lancer le programme. Si vous décidez de rendre le programme disponible uniquement en ligne, il est téléchargé chaque fois qu'un utilisateur souhaite l'exécuter ; il n'est pas ajouté au menu **Démarrer**.

Essayez !

Pour publier sur un serveur Web

1. Dans le menu **Fichier**, sélectionnez **Ouvrir un projet**.
2. Dans la boîte de dialogue **Ouvrir un projet**, naviguez jusqu'à un projet **Application Windows**, puis cliquez sur **Ouvrir**.
3. Appuyez sur F5 pour exécuter le projet. Vous devez résoudre les erreurs éventuelles avant de poursuivre.
4. Dans le menu **Déboguer**, choisissez **Arrêter le débogage**.
5. Dans le menu **Générer**, sélectionnez **Publier NomProjet**, où **NomProjet** est le nom de votre projet.

   L'**Assistant Publication** s'exécute.
6. Dans la page **Où souhaitez-vous publier l'application ?** de l'**Assistant Publication**, entrez l'URL du site Web où vous souhaitez publier votre programme. Par exemple, `http://www.microsoft.com/myprogram`.

Remarque
Pour publier votre programme sur un serveur Web, ce dernier doit exécuter IIS (Internet Information Services), les extensions FrontPage doivent être installées, et vous devez disposer de privilèges d'administration dans IIS.

   Cliquez sur **Suivant** pour passer à la page suivante de l'Assistant.
7. Sur la page **L'application sera-t-elle disponible hors connexion ?**, sélectionnez **Oui, cette application est disponible en ligne ou hors connexion**, la valeur par défaut.
8. Cliquez sur **Terminer** pour publier le programme.

   Le programme est publié sur le site Web spécifié, et une page HTML est créée.
9. Sur un autre ordinateur, ouvrez Internet Explorer, naviguez jusqu'à l'URL saisie auparavant, puis cliquez sur le lien **Installer** pour installer le programme.

Étapes suivantes

Dans cette leçon, vous avez appris comment publier un programme sur un site Web à l'aide de la publication ClickOnce. À ce stade, différentes possibilités s'offrent à vous.

Si vous n'étiez pas en mesure de graver un CD-ROM ou un DVD-ROM, consultez
Envoi de votre programme par messagerie électronique : création d'un fichier compressé.

Si vous souhaitez apprendre comment intégrer des composants requis à votre programme, consultez
Présentation détaillée : ajout de composants requis dans votre programme.

Sinon, vous avez terminé les leçons de la visite guidée. Vous pouvez passer à la rubrique suivante pour consulter quelques suggestions quant à d'autres ressources pédagogiques relatives à Visual Basic.

Leçon suivante : Étape suivante : que faire ensuite ?

Voir aussi
Concepts
Partage de votre programme : introduction au déploiement

Envoi de votre programme par messagerie électronique : création d'un fichier compressé

Dans cette leçon, vous apprendrez comment intégrer votre programme sous la forme d'un fichier compressé (zippé) et l'envoyer par courrier électronique.

Une fois votre programme terminé, vous pouvez souhaiter partager le fruit de votre dur labeur avec vos amis et collègues. Le meilleur moyen consiste à publier votre programme sur un CD-ROM ou un DVD-ROM à l'aide de la publication ClickOnce. Si vous n'avez pas accès à un graveur de CD-ROM ou de DVD-ROM, peut-être êtes-vous en mesure de distribuer votre programme en l'intégrant à un fichier compressé et en l'envoyant par courrier électronique.

Remarque

Vous ne devez jamais ouvrir une pièce jointe de quelque type que ce soit, y compris un fichier .zip, émanant d'un expéditeur en qui vous n'avez pas confiance. Les pièces jointes non souhaitées ou non sollicitées peuvent contenir des virus susceptibles de bloquer ou d'infiltrer votre ordinateur, ou de voler vos informations personnelles. Pour des raisons similaires, vous ne dev ez pas envoyer votre application à un utilisateur qui ne l'attend pas.

Pour la plupart des programmes simples, vous pouvez intégrer le fichier exécutable (*.exe*) et le fichier de configuration (*.config*) à un fichier compressé et le distribuer. Par exemple, si vous possédez un programme simple nommé *Calculator*, vous aurez besoin des fichiers *Calculator.exe* et *Calculator.exe.config*, placés dans le répertoire *your Visual Studio Projects/Calculator/Calculator/bin* , où *your Visual Studio Projects* désigne le répertoire dans lequel sont enregistrés vos projets Visual Studio, généralement *My Documents/Visual Studio/Projects*.

Remarque

Tous les programmes créés avec Visual Basic nécessitent Microsoft .NET Framework version 2.0. Si vous distribuez votre pro gramme par courrier électronique, assurez-vous que le .NET Framework est déjà installé sur les ordinateurs des destinataires .

Conseil

Le .NET Framework est distribué avec beaucoup d'autres programmes ; il peut également être téléchargé sur le site Web de Microsoft. Pour vérifier si le .NET Framework est installé, ouvrez le **Panneau de configuration**, puis cliquez sur **Ajout/Sup pression de programmes**. Ensuite, recherchez Microsoft .NET Framework version 2.0.

Pour les programmes plus sophistiqués, notamment ceux qui incluent une base de données locale, vous pouvez être amené à inclure des fichiers supplémentaires. Dans ce cas, la meilleure approche consiste à publier votre programme à l'aide de la technologie ClickOnce, plutôt que de l'envoyer par courrier électronique.

Essayez !

La procédure suivante montre comment compresser un fichier à l'aide de la fonctionnalité Dossier Compressé par défaut de Windows. Si vous avez installé un programme différent pour travailler avec les fichiers .zip, les étapes nécessaires à leur compression peuvent être différentes.

Pour envoyer un programme simple par messagerie électronique

1. Sélectionnez un projet qui vous avez terminé et exécuté, puis dans l'Explorateur Windows, naviguez jusqu'au dossier du projet.
2. Dans le dossier du projet, ouvrez le dossier appelé Bin, puis sélectionnez le dossier Debug.
3. Sélectionnez le fichier *ProjectName*.exe (où *ProjectName* est le nom de votre projet), cliquez avec le bouton droit sur le fichier et choisissez **Envoyer vers**, **Dossier compressé**.

   Un nouveau fichier nommé *ProjectName*.zip est créé, puis ajouté au répertoire. Vous pouvez envoyer ce fichier en tant que pièce jointe à un courrier électronique.

Étapes suivantes

Dans cette leçon, vous avez appris comment intégrer votre programme pour l'envoyer par courrier électronique. Dans la leçon suivante, vous apprendrez comment extraire et exécuter un programme que vous avez reçu.

Leçon suivante : Obtention d'un programme par messagerie électronique : extraction d'un programme compressé

Voir aussi

Tâches

Distribution de votre programme sur CD : publication avec ClickOnce

Concepts

Partage de votre programme : introduction au déploiement

Autres ressources

Distribution de votre programme

Obtention d'un programme par messagerie électronique : extraction d'un programme compressé

Dans cette leçon, vous apprendrez comment extraire et exécuter un programme reçu d'un ami ou d'un collègue par courrier électronique.

Vos amis et collègues peuvent parfois souhaiter vous faire partager le fruit de leur dur labeur. Dans la leçon précédente, vous avez appris comment ajouter votre programme à un fichier compressé (compressé) afin qu'il puisse être envoyé par courrier électronique. Dans cette leçon, vous apprendrez comment ouvrir et exécuter un programme reçu par courrier électronique.

Remarque

Vous ne devez jamais ouvrir une pièce jointe de quelque type que ce soit, y compris un fichier .zip, émanant d'un expéditeur en qui vous n'avez pas confiance. Les pièces jointes non souhaitées ou non sollicitées peuvent contenir des virus susceptibles de bloquer ou d'infiltrer votre ordinateur, ou de voler vos informations personnelles. Pour les mêmes raisons, vous ne devez pas envoyer votre application à un utilisateur qui ne l'attend pas.

Essayez !

Pour extraire un programme compressé (zippé)

1. Naviguez jusqu'au fichier .zip que vous avez créé dans la leçon précédente. Si vous n'avez pas terminé cette leçon, prenez un moment pour finir Envoi de votre programme par messagerie électronique : création d'un fichier compressé.
2. Cliquez avec le bouton droit sur le fichier .zip, puis sélectionnez **Extraire tout**.

   L'**Assistant Extraction** s'affiche.
3. Dans l'Assistant, cliquez sur le bouton **Suivant**.

   Le volet **Sélectionner une destination** s'affiche.
4. Tapez le chemin d'accès au dossier dans lequel vous souhaitez installer le programme, ou cliquez sur le bouton **Parcourir** pour naviguer jusqu'à ce dossier, puis cliquez sur **Suivant**.

   Les fichiers sont extraits et le volet **Extraction terminée** s'affiche.
5. Activez la case à cocher **Afficher les fichiers extraits**, puis cliquez sur **Terminer**.

   Une nouvelle fenêtre de l'Explorateur Windows s'ouvre et présente le dossier choisi à l'étape précédente.
6. Double-cliquez sur le fichier *ProgramName*.exe (où *ProgramName* est le nom du programme) pour exécuter le programme.

   Remarque

   Tous les programmes créés avec Visual Basic nécessitent le Microsoft .NET Framework version 2.0. Si vous distribuez v otre programme par courrier électronique, assurez-vous que le .NET Framework est déjà installé sur les ordinateurs de s destinataires.

   Conseil

   Le .NET Framework est distribué avec de nombreux autres programmes. Il peut également être téléchargé à partir du si te Web Microsoft. Pour vérifier si le .NET Framework est installé, ouvrez le **Panneau de configuration**, puis cliquez su r **Ajout/Suppression de programmes**. Ensuite, recherchez Microsoft .NET Framework version 2.0.

Étapes suivantes

Dans cette leçon, vous avez appris comment extraire et exécuter un programme reçu par courrier électronique d'une source fiable. Vous avez également appris qu'il était dangereux d'exécuter un fichier provenant d'une source à laquelle vous ne faites pas confiance. Vous avez maintenant terminé la présentation des notions de base ! Vous pouvez prendre le temps d'explorer seul Visual Basic et d'écrire quelques programmes. Vous pouvez également lire différentes suggestions pour en savoir plus sur Visual Basic.

Leçon suivante : Étape suivante : que faire ensuite ?

Voir aussi

Tâches

Envoi de votre programme par messagerie électronique : création d'un fichier compressé

Autres ressources

Distribution de votre programme

Visite guidée de Visual Basic

Étape suivante : que faire ensuite ?

Vous avez terminé les leçons Visite guidée de Visual Basic. Si vous n'êtes toutefois pas encore un expert Visual Basic, vous en savez assez pour commencer à écrire vos propres programmes.

Si vous avez déjà une idée de programme, lancez-vous et commencez. Si vous souhaitez d'abord en savoir plus sur Visual Basic, consultez ce qui suit.

Pour en savoir plus

Augmentation de la productivité : développement rapide d'application

Lors de l'avènement de la programmation informatique, l'achèvement d'un programme simple pouvait durer des jours, voire des semaines.

Conseils et astuces : je ne savais pas que c'était possible

Dans la Visite guidée de Visual Basic, vous avez appris comment effectuer différentes opérations, mais il ne s'agissait là que d'un échantillon des actions réalisables à l'aide de Visual Basic 2005.

Ressources communautaires : l'union fait la force

L'apprentissage de la programmation n'est pas toujours un jeu d'enfant ; même les programmeurs les plus expérimentés rencontrent des problèmes qu'ils ne peuvent pas résoudre eux-mêmes.

Informations complémentaires : ressources pour l'apprentissage de Visual Basic

Lorsque vous aurez terminé toutes les leçons de la Visite guidée de Visual Basic, vous en aurez appris suffisamment pour commencer à écrire vos propres programmes, mais vous ne serez certainement pas un expert.

Voler n'est pas un crime : exemples et Starter Kits

Écrire du code c'est bien, mais ne pas en écrire, c'est mieux encore. Visual Basic 2005 facilite la réutilisation du code contenu dans les rubriques d'aide, les exemples d'applications et les Starter Kits.

Plus en profondeur : je souhaite en savoir davantage sur...

La Visite guidée de Visual Basic vous a présenté certaines opérations réalisables à l'aide de Visual Basic Express et de Visual Basic 2005, mais il ne s'agit que de la partie visible de l'iceberg. Si vous souhaitez approfondir la matière abordée dans la visite guidée, commencez ici.

Voir aussi

Tâches

Dépannage de Visual Basic Express

Concepts

Comment faire dans Visual Basic Express

Autres ressources

Distribution de votre programme

Visite guidée de Visual Basic

Augmentation de la productivité : développement rapide d'application

Dans les premières années de la programmation informatique, l'achèvement d'un programme simple pouvait durer des jours, voire des semaines. Lorsque Visual Basic est apparu pour la première fois en 1991, il a révolutionné la programmation. Il n'était désormais plus nécessaire d'écrire du code pour créer une interface utilisateur ou de se soucier de la gestion de la mémoire. Ce nouveau type de programmation fut appelé *développement rapide d'application* ou RAD (Rapid Application Development).

Le principal avantage de la programmation RAD est l'augmentation de la productivité. En effet, Visual Basic 2005 propose un grand nombre de fonctionnalités qui peuvent vous aider à créer de meilleures applications plus rapidement que jamais. Vous trouverez ci-dessous quelques-unes de ces fonctionnalités.

Remarque

Si vous utilisez Visual Basic Express, certains des liens d'aide fournis sur cette page risquent de ne pas être disponibles, selon les options que vous avez choisies pendant l'installation. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Extraits de code

Une façon d'augmenter votre productivité consiste à éviter d'écrire le même code à de nombreuses reprises. Visual Basic 2005 comprend une bibliothèque de code contenant environ 500 extraits de code, appelés *extraits de code IntelliSense*, qui sont prêts à être insérés dans votre application. Chaque extrait de code exécute une tâche de programmation complète, telle que la création d'un fichier, l'envoi d'un message électronique ou le dessin d'un cercle. Quelques clics de souris suffisent pour insérer un extrait dans votre code source.

Une fois l'extrait de code inséré, les parties du code qui doivent être remplacées sont mises en surbrillance ; vous pouvez entrer vos propres valeurs le cas échéant. Par exemple, un extrait de code pour dessiner une ligne sur un formulaire possède des valeurs pour la couleur, l'emplacement et la longueur de la ligne. Vous pouvez modifier ces valeurs en fonction de vos besoins, ou ne rien faire et dessiner une ligne à l'aide des valeurs par défaut.

Vous pouvez également créer vos propres extraits de code adaptés à vos propres besoins, les ajouter à la bibliothèque, puis les utiliser lorsque vous en avez besoin. Lorsque vous créez vos propres extraits de code, vous devez déterminer les parties du code qui doivent être mises en surbrillance et spécifier les valeurs par défaut. Pour plus d'informations, consultez Création et utilisation d'extraits de code IntelliSense.

Citons la lecture et l'écriture de texte dans un fichier comme tâches courantes pouvant être réalisées à l'aide d'extraits de code. La procédure suivante montre comment les extraits de code peuvent améliorer votre productivité.

Essayez !

Pour utiliser des extraits de code

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans le volet **Modèles** de la boîte de dialogue **Nouveau projet**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez `Snippets`, puis cliquez sur **OK**.

   Un nouveau projet Windows Forms s'ouvre.
4. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code.
5. Dans l'éditeur de code, cliquez avec le bouton droit sur le gestionnaire d'événements `Form1_Load` et choisissez **Insérer un extrait** dans le menu déroulant.

   La liste des catégories d'extraits s'affiche.
6. Double-cliquez sur **Traitement de lecteurs, de dossiers et de fichiers**.

   La liste des extraits de code s'affiche.
7. Double-cliquez sur **Écrire le texte dans un fichier**.

Le code suivant est inséré et les éléments `"C\Test.txt"` ainsi que `"Text"` sont mis en surbrillance.

```
My.Computer.FileSystem.WriteAllText("C:\Test.txt", "Text", True)
```

Remarque

La méthode **WriteAllText** crée le fichier s'il n'existe pas. S'il existe déjà, elle ajoute le texte à la fin du fichier.

8. Remplacez `"C\Test.txt"` par `"C\MySnippetTest.txt"` et remplacez `"Text"` par `"This is really fast!"`.
9. Ajoutez un deuxième extrait de code. Cliquez avec le bouton droit et choisissez **Insérer un extrait** dans le menu.
10. Double-cliquez sur **Traitement de lecteurs, de dossiers et de fichiers**.
11. Double-cliquez sur **Lire le texte d'un fichier**.

    Le code suivant est inséré et `"C\Test.txt"` est mis en surbrillance.

```
Dim fileContents As String
fileContents = My.Computer.FileSystem.ReadAllText("C:\Test.txt")
```

12. Remplacez `"C\Test.txt"` par `"C\MySnippetTest.txt"`.
13. Ajoutez le code suivant sous le dernier extrait de code pour afficher le résultat.

```
MsgBox(fileContents)
```

14. Appuyez sur F5 pour exécuter le programme.

    Un fichier contenant le texte spécifié est créé et un message s'affiche avec le contenu du fichier.

Prenez le temps de vous familiariser avec les extraits de code fournis dans Visual Basic 2005 ; ils peuvent vous faire gagner du temps et économiser beaucoup d'efforts lors de l'écriture du code. Pour plus d'informations, consultez Comment : gérer des extraits de code.

Développement avec My

Une autre fonctionnalité RAD introduite dans Visual Basic 2005 porte le nom de **My**. **My** est un jeu d'objets qui contiennent des fonctions utilisées couramment liées à l'ordinateur, à l'application, à l'utilisateur, etc. Vous pouvez considérer **My** comme un raccourci pour accéder à des fonctions qui exigeraient normalement une grande quantité de code supplémentaire.

Par exemple, supposons que vous souhaitiez déterminer le numéro de version de votre application. Dans la version précédente de Visual Basic, le code se serait présenté comme suit.

```
Dim VersionNumber As String
VersionNumber = System.Diagnostics.FileVersionInfo.GetVersionInfo _ (System.Reflection.Asse
mbly.GetExecutingAssembly.Location).FileVersion
```

À l'aide du nouvel objet **My.Application**, il se présente comme suit.

```
Dim VersionNumber As String
VersionNumber = My.Application.Info.Version.ToString
```

Comme vous pouvez le constater, **My** est beaucoup plus simple (et beaucoup plus convivial), et permet de gagner du temps tout en économisant des efforts. Vous pourriez toujours utiliser l'autre manière pour déterminer le numéro de version, mais pourquoi le feriez-vous ?

Vous ne vous en êtes peut-être pas rendu compte, mais vous avez déjà utilisé **My** dans plusieurs leçons précédentes. Lorsque vous entrerez du code pour votre application suivante, explorez les objets **My** en tapant `My` et en parcourant la liste des éléments qui s'affichent. Pour plus d'informations, consultez Développement avec My.

IntelliSense

Lorsque vous avez entré du code au cours des leçons précédentes, vous avez probablement remarqué qu'au fur et à mesure que vous tapiez, une liste déroulante de choix s'affichait dans l'éditeur de code. Il s'agit d'un exemple de la fonctionnalité appelée IntelliSense.

IntelliSense propose une série de fonctionnalités qui facilitent l'accès aux guides de référence du langage. Lors de l'écriture de code, il n'est pas nécessaire de laisser l'éditeur de code rechercher des informations sur les éléments de langage. Vous pouvez rester où vous êtes, rechercher les informations dont vous avez besoin, insérer directement les éléments de langage dans votre code, et même demander à IntelliSense de saisir les informations à votre place.

IntelliSense est également utile lors du débogage. Dans l'éditeur de code, vous pouvez déplacer le curseur sur une variable contenue dans votre code afin d'afficher une info-bulle avec la valeur actuelle de la variable. IntelliSense est également disponible lors de la saisie de code dans la fenêtre **Exécution**. Pour plus d'informations, consultez Utilisation de la fonctionnalité IntelliSense.

Étapes suivantes

Dans cette leçon, vous avez appris quelques-unes des fonctionnalités de productivité disponibles dans Visual Basic 2005. De nombreuses autres fonctionnalités font de Visual Basic 2005 l'outil idéal pour le développement rapide d'applications. Vous trouverez ci-dessous des liens vers certaines de celles-ci.

Dans la leçon suivante, vous découvrirez certaines possibilités surprenantes de Visual Basic 2005.

Leçon suivante : Conseils et astuces : je ne savais pas que c'était possible.

Voir aussi

Concepts

Étape suivante : que faire ensuite ?

Comment faire dans Visual Basic Express

Autres ressources

Distribution de votre programme

Visite guidée de Visual Basic

Conseils et astuces : je ne savais pas que c'était possible

Dans la Visite guidée de Visual Basic, vous avez appris un certain nombre de choses, mais il ne s'agissait là que d'un aperçu de ce que vous pouvez faire avec Visual Basic 2005. Même les soi-disant experts en Visual Basic apprennent constamment de nouveaux conseils et de nouvelles astuces. Vous trouverez ci-dessous quelques exemples de possibilités moins connues.

Remarque

Si vous utilisez Visual Basic Express, certains des liens d'aide fournis sur cette page risquent de ne pas être disponibles, selon les options que vous avez choisies pendant l'installation. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Créer des Windows Forms avec une forme spécifique

Vous en avez assez des formulaires rectangulaires ? Vous souhaitez donner à votre application une apparence, comme au Lecteur Windows Media ? C'est un jeu d'enfant grâce à Visual Basic 2005 : vous pouvez créer une image bitmap de la forme souhaitée et l'utiliser comme formulaire, en ajoutant le code afin que le formulaire puisse être déplacé et refermé. Pour plus d'informations, consultez Comment : créer des Windows Forms non rectangulaires.

Créer des fenêtres fractionnées

Vous souhaitez créer un formulaire semblable au Document Explorer que vous utilisez, avec plusieurs zones qui peuvent être redimensionnées par l'utilisateur ? Le contrôle Windows Forms **SplitContainer** permet d'effectuer cette opération sans code. Il vous suffit de faire glisser un contrôle **SplitContainer** jusqu'à votre formulaire, puis d'y ajouter d'autres contrôles ; le comportement de redimensionnement est automatiquement disponible lorsque vous lancez l'application.

Vous pouvez également ajouter plusieurs contrôles **SplitContainer** au formulaire pour disposer de zones redimensionnables au sein d'autres zones, et créer ainsi une application qui ressemble à Microsoft Outlook. Pour plus d'informations, consultez SplitContainer, contrôle (Windows Forms).

Lire des sons

Si vous créez un jeu, vous souhaiterez probablement que votre programme émette des sons en réponse à différents événements. L'objet **My.Computer.Audio** permet d'effectuer cette opération, de lire des fichiers .wav que vous incluez dans votre application ou même de lire directement des fichiers à partir d'Internet. Pour plus d'informations, consultez My.Computer.Audio, objet.

Enregistrer les préférences utilisateur

Vous avez probablement remarqué qu'un grand nombre d'applications Windows « mémorisent » vos préférences, telles que la manière dont vous aimez organiser vos fenêtres ou les barres d'outils qui étaient affichées la dernière fois que vous avez utilisé l'application. Vous pouvez effectuer la même opération dans vos propres programmes en créant et utilisant des *paramètres d'application* pour stocker des informations et les récupérer lors de la prochaine exécution de l'application. Pour plus d'informations, consultez Vue d'ensemble des paramètres d'application.

Ajouter la personnalisation

Vous êtes-vous jamais demandé comment les pages Web se souviennent du nom d'un utilisateur, en affichant des messages, tels que "Welcome back (*insert your name here*)" ? Vous pouvez effectuer une opération similaire dans votre propre application en utilisant la propriété My.User.Name pour obtenir le nom de l'utilisateur actuellement connecté à l'ordinateur. Pour plus d'informations, consultez My.User, objet.

Utiliser du code Visual Basic 6.0

Possédez-vous un exemple de code Visual Basic 6.0 que vous souhaiteriez utiliser dans Visual Basic 2005 ? L'outil **Mettre à niveau le code Visual Basic 6** peut convertir le code Visual Basic 6.0 et l'insérer dans votre code Visual Basic 2005. Si le code ne peut pas être complètement converti, des commentaires sont ajoutés avec des liens vers des rubriques d'aide expliquant les actions à exécuter pour que le code fonctionne. Pour plus d'informations, consultez Comment : mettre à niveau du code Visual Basic 6.0 à l'aide de la boîte de dialogue Mettre à niveau le code Visual Basic 6.

Utiliser le multithreading pour améliorer les performances

Les applications Visual Basic peuvent effectuer simultanément plusieurs tâches à l'aide d'une technique appelée *multithreading*. Le multithreading est un processus dans lequel une tâche s'exécute sur un thread d'exécution distinct, ce qui améliore les performances et la sensibilité de votre programme.

Par exemple, supposons que vous disposiez d'un programme qui télécharge un fichier à partir d'Internet. Le téléchargement peut durer un certain temps, empêchant ainsi l'utilisateur d'effectuer d'autres opérations jusqu'à ce qu'il soit terminé. En cas d'exécution du téléchargement sur un thread distinct, l'utilisateur peut se consacrer librement à d'autres opérations pendant que le fichier se télécharge en arrière-plan. Pour plus d'informations, consultez Multithreading dans Visual Basic.

Visual Basic 2005 possède également un composant **BackgroundWorker** qui facilite la réalisation d'une tâche en arrière-plan. Pour plus d'informations, consultez
Procédure pas à pas : implémentation d'un formulaire qui utilise une opération d'arrière-plan.

Créer la documentation XML

La documentation XML permet d'ajouter des commentaires à une classe ou à un contrôle utilisateur pour expliquer à un autre programmeur comment les utiliser. Par exemple, supposons que votre contrôle utilisateur possède une propriété nommée "Stretch" (le nom de la propriété n'indique pas sa véritable fonction). La documentation XML permet d'ajouter une description, telle que "Détermine si le texte doit être étiré pour remplir la bannière" ; cette description s'affiche dans la fenêtre **Propriétés** et dans IntelliSense. Pour plus d'informations, consultez Documentation de votre code avec le langage XML (Visual Basic).

Installer le .NET Framework avec votre programme

Tous les programmes que vous créez avec Visual Basic 2005 exigent que le runtime .NET Framework soit déjà installé sur tout ordinateur sur lequel vous souhaitez les exécuter ; certains programmes peuvent également exiger d'autres fichiers ou des *composants requis*. Si vous partagez votre programme à l'aide de la publication ClickOnce, vous pouvez utiliser une fonctionnalité appelée *amorçage* pour inclure ces fichiers et les installer automatiquement, le cas échéant. Pour plus d'informations, consultez Comment : installer les composants requis avec une application ClickOnce.

Étapes suivantes

Dans cette leçon, vous avez appris certaines des opérations réalisables à l'aide de Visual Basic 2005. Bien sûr, il ne s'agissait que d'un petit aperçu. Au fur et à mesure que vous utiliserez Visual Basic 2005, vous découvrirez beaucoup d'autres actions que vous n'imaginiez pas pouvoir réaliser.

Dans la leçon suivante, vous apprendrez à vous connecter avec d'autres utilisateurs de Visual Basic 2005.

Leçon suivante : Ressources communautaires : l'union fait la force

Voir aussi

Tâches

Augmentation de la productivité : développement rapide d'application

Concepts

Étape suivante : que faire ensuite ?

Comment faire dans Visual Basic Express

Autres ressources

Visite guidée de Visual Basic

Ressources communautaires : l'union fait la force

L'apprentissage de la programmation n'est pas toujours un jeu d'enfant ; même les programmeurs les plus expérimentés rencontrent des problèmes qu'ils ne peuvent pas résoudre eux-mêmes. Heureusement, il existe plusieurs millions d'autres programmeurs de par le monde qui utilisent Visual Basic, et la plupart d'entre eux n'hésitent pas à s'aider mutuellement.

Remarque

Si vous utilisez Visual Basic Express, certains des liens d'aide fournis sur cette page risquent de ne pas être disponibles, selon les options que vous avez choisies pendant l'installation. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Utilisation de forums pour échanger des informations

L'un des meilleurs moyens d'obtenir de l'aide supplémentaire avec Visual Basic consiste à visiter un *forum en ligne*, une communauté virtuelle où les utilisateurs posent des questions et répondent à celles des autres. Vous avez peut-être remarqué la présence du bouton **Poser une question** dans la barre d'outils de l'afficheur d'aide. Vous pouvez cliquer sur ce bouton pour accéder à la page de recherche des forums de communauté MSDN. Vous pouvez saisir une question et lancer une recherche qui retourne toute réponse précédente en rapport avec votre question.

Remarque

Vous devez disposer d'une connexion active à Internet, et tout serveur proxy ou pare-feu doit être configuré pour autoriser l'accès aux forums de communauté MSDN.

Une question ou une réponse individuelle dans un forum est appelée *publication* ; une collection de publications sur un même sujet est également connue sous le nom de *thread*. Si vous ne trouvez pas de thread précédent répondant à votre question, vous pouvez publier un message sur le forum avec votre question spécifique, puis vérifier ultérieurement le thread pour déterminer si quelqu'un y a répondu.

Astuces pour l'utilisation de forums :

- Lisez d'abord la documentation Visual Basic. Si une rubrique est abordée dans l'aide, ne vous attendez pas à ce que d'autres utilisateurs la recherche à votre place. En revanche, si de l'aide est disponible à ce sujet, mais si vous ne la comprenez pas, vous pouvez demander des explications.
- Parcourez le forum pour déterminer si la question a déjà été posée et a déjà reçu une réponse. Si vous publiez une question fréquemment posée, votre publication risque d'être ignorée.
- Choisissez le forum correct. Si vous publiez une question relative au langage Visual Basic dans un forum Windows Forms, il est moins probable que vous obteniez une réponse que si vous la publiez sur le forum du langage Visual Basic.
- Utilisez un titre descriptif pour votre publication. Si vous demandez « Quelqu'un peut-il m'aider ? », vous êtes moins susceptible de recevoir une réponse que si vous demandez « Quelqu'un sait-il comment créer un contrôle utilisateur avec un arrière-plan transparent ? ».
- Exprimez clairement votre question. Expliquez ce que vous essayez de faire et ce qui ne fonctionne pas ; indiquez ce que vous avez déjà essayé et quels étaient les résultats.
- Ajoutez le code le cas échéant. Si une ligne particulière de code provoque une erreur, incluez suffisamment de code pour indiquer clairement l'opération que cette ligne tente d'exécuter. Toutefois, n'essayez pas de publier l'intégralité de votre application et d'attendre que quelqu'un la débogue à votre place.
- Si vous posez une question, essayez de répondre à celle d'un autre utilisateur. Plus vous êtes disposé à aider les autres membre de la communauté, plus ils seront disposés à vous aider.
- Soyez toujours poli. Si quelqu'un répond à votre question, remerciez-le. N'oubliez pas que quiconque répond aux questions sur le forum le fait volontairement.

Outre les forums de communauté MSDN, il existe de nombreux autres forums liés à Visual Basic, aux Windows Forms et aux autres technologies. Vous pouvez accéder à certains en choisissant **Communauté Code Wise** dans le menu **Communauté** de Visual Basic 2005, ou vous pouvez les rechercher en ligne.

Essayez !

Pour visiter un forum

1. Dans le menu **Communauté** de Visual Basic 2005, sélectionnez **Poser une question**.

   La page de recherche des forums de communauté MSDN s'affiche.

2. Dans le champ **Rechercher**, tapez `VB Transparent User Control`, puis cliquez sur **OK**.

   S'il existe des publications précédentes correspondant aux mots clés, elles s'affichent.

3. Examinez les publications pour déterminer si certaines d'entre elles expliquent comment créer un contrôle utilisateur transparent.

   Si aucune des publications ne répond à votre question, vous pouvez publier une question plus spécifique en choisissant **Généralités sur Visual Basic** dans la liste **Réponse introuvable ?**.

4. Pour vérifier si votre question a reçu une réponse, choisissez **Vérifier le statut des questions** dans le menu **Communauté**.

   Conseil

   Vous pouvez également choisir d'être averti par messagerie électronique, par messagerie instantanée ou même par m essage texte sur votre téléphone portable lorsqu'un thread reçoit une réponse. Lorsque vous entrez une publication ou répondez à une publication sur un forum, il vous suffit de cliquer sur le bouton **M'avertir**.

Autre ressources de communauté

Outre les forums en ligne, il existe beaucoup d'autres ressources de communauté qui peuvent vous aider à apprendre à utiliser Visual Basic 2005.

La **page de démarrage** qui s'affiche lorsque vous démarrez Visual Basic Express contient une section **Informations** contenant des liens vers des événements de communauté, tels que des conversations en ligne et des webdiffusions, qui sont mis à la disposition de tous. Ces liens sont régulièrement mis à jour.

Conseil

Les liens de la **Page de démarrage** sont fournis par un flux RSS ; vous pouvez choisir les types des liens que vous souhaitez afficher. Pour plus d'informations, consultez Comment : personnaliser la section Informations de la page de démarrage.

La fonction Rechercher de l'aide permet d'effectuer une recherche non seulement dans le système d'aide local, mais aussi dans MSDN Online Library et dans la Communauté Code Wise (collection de ressources en ligne pour Visual Basic). MSDN Online Library peut contenir des versions plus récentes des rubriques d'aide, ainsi que des articles qui ne sont pas présents dans la documentation ; les sites Web qui constituent la Communauté Code Wise fournissent des informations utiles émanant de nombreux d'experts de Visual Basic.

Remarque

Pour accéder à MSDN Online Library et à la Communauté Code Wise, vous devez disposer d'une connexion active à Internet, et tout serveur proxy ou pare-feu doit être configuré pour y autoriser l'accès. De plus, vos paramètres d'aide doivent être con figurés pour l'accès en ligne. Pour plus d'informations, consultez
Comment : inclure du contenu en ligne dans les résultats de recherche et de l'aide F1.

L'équipe Visual Basic de Microsoft est toujours intéressée par vos commentaires, que vous souhaitiez proposer une amélioration à apporter à Visual Basic 2005, signaler un bogue de produit ou formuler des suggestions relatives à la documentation.

Le Product Feedback Center permet tant de formuler des suggestions sur le produit et la documentation, que de signaler des bogues ou consulter les informations fournies par des tiers. Vous pouvez également suivre la progression de vos suggestions et rapports de bogues, voter pour ou contre des fonctionnalités suggérées, etc. Pour accéder au Product Feedback Center, choisissez **Envoyer des commentaires** dans le menu **Communauté**.

Le Centre pour développeurs Microsoft Visual Basic contient des articles fréquemment mis à jour et d'autres informations à propos de Visual Basic. Pour accéder au Centre pour développeurs, choisissez **Centre pour développeurs** dans le menu **Communauté**.

Le site CodeWise Community est une ressource permettant d'obtenir des astuces de tiers, des exemples de code, des conseils et des informations émanant d'experts indépendants de Microsoft .NET Framework et de Microsoft Visual Studio 2005. Pour accéder au site CodeWise Community, choisissez **Communauté Code Wise** dans le menu **Communauté**.

Les *groupes d'utilisateurs* sont des organisations locales de programmeurs possédant des intérêts semblables. Il existe dans le monde entier de nombreux groupes d'utilisateurs consacrés à Visual Basic et à Visual Basic 2005. La plupart de ces groupes se rencontrent régulièrement pour montrer des techniques de codage, échanger des informations et établir des contacts avec d'autres programmeurs. Si vous vivez dans une grande ville ou à proximité, il existe probablement un groupe d'utilisateurs Visual Basic près de chez vous.

Étapes suivantes

Dans cette leçon, vous avez appris quelques-unes des ressources mises à la disposition de communauté Visual Basic internationale. Dans la leçon suivante, vous apprendrez les ressources permettant d'en savoir davantage sur Visual Basic 2005.

Leçon suivante : Informations complémentaires : ressources pour l'apprentissage de Visual Basic.

Voir aussi

Tâches

Conseils et astuces : je ne savais pas que c'était possible

Concepts

Étape suivante : que faire ensuite ?

Comment faire dans Visual Basic Express

Autres ressources

Visite guidée de Visual Basic

Informations complémentaires : ressources pour l'apprentissage de Visual Basic

Lorsque vous aurez terminé toutes les leçons de la Visite guidée de Visual Basic, vous en aurez appris suffisamment pour commencer à écrire vos propres programmes, mais vous ne serez certainement pas un expert. Même les professionnels expérimentés de Visual Basic apprennent en permanence ; personne ne peut affirmer tout connaître de Visual Basic 2005. Vous trouverez ci-dessous quelques suggestions pour en apprendre davantage sur Visual Basic.

Remarque

Si vous utilisez Visual Basic Express, certains des liens d'aide fournis sur cette page risquent de ne pas être disponibles, selon les options que vous avez choisies pendant l'installation. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Obtention d'aide pendant votre travail

Pour bon nombre de gens, la pratique constitue le meilleur apprentissage. Le moyen le plus facile d'obtenir de l'aide lors de l'utilisation de l'environnement de développement intégré (IDE) Visual Basic Express consiste à appuyer sur la touche F1 pour afficher l'aide *contextuelle*.

- Appuyez sur F1 lorsqu'une fenêtre de l'IDE est sélectionnée pour afficher une rubrique d'aide qui décrit la fenêtre et la manière de l'utiliser.
- Appuyez sur F1 en mode Design lorsqu'un contrôle est sélectionné pour afficher une rubrique d'aide relative à ce contrôle.
- Appuyez sur F1 dans la fenêtre **Propriétés** pour afficher l'aide relative à la propriété sélectionnée.
- Appuyez sur F1 dans l'éditeur de code pour afficher la rubrique de référence du langage relative au mot clé de langage le plus proche de la position actuelle du curseur.

Essayez !

Pour afficher l'aide contextuelle.

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, dans le volet **Modèles**, cliquez sur **Application Windows**.
3. Sélectionnez le formulaire, et dans la fenêtre **Propriétés**, sélectionnez la propriété **KeyPreview**.
4. Appuyez sur F1.

   Une rubrique d'aide décrivant la propriété **KeyPreview** s'affiche.

   Conseil

   Vous avez pu constater que certaines rubriques d'aide comprennent des exemples de syntaxe et de code de langages autres que Visual Basic. Si vous souhaitez n'afficher que la syntaxe et le code Visual Basic, pointez sur le bouton **Filtre de langage** situé en haut de votre rubrique d'aide ; dans la liste déroulante, désactivez les cases à cocher correspondant à toutes les entrées, à l'exception de **Visual Basic (Utilisation)**. Votre paramètre sera mémorisé jusqu'à ce que vous le modifiiez à nouveau.

5. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code et entrez les informations suivantes dans le gestionnaire d'événements **Form1_Load**.

   ```
   Dim CurrentUser As String
   CurrentUser = My.User.Name
   ```

6. Placez le curseur à l'intérieur du mot `Name` et appuyez sur F1.

   Une rubrique d'aide correspondant à la propriété **My.User.Name** s'affiche.

7. Placez le curseur à l'intérieur du mot `User` et appuyez à nouveau sur F1.

   Une rubrique d'aide correspondant à l'objet **My.User** s'affiche.

Documentation du produit

La documentation produit de Visual Basic 2005 contient une multitude d'informations pour l'apprentissage de Visual Basic, notamment de nombreuses rubriques « Comment », procédures pas à pas et informations d'aide conceptuelle détaillées, ainsi que des rubriques de référence pour chaque membre du .NET Framework. Il est conseillé de prendre le temps de parcourir la documentation pour en savoir davantage sur les fonctionnalités de Visual Basic 2005 dont vous ne connaissez peut-être pas l'existence.

Selon les choix que vous avez effectués lors de l'installation de Visual Basic Express, il se peut que certaines rubriques ne soient pas disponibles. Les fichiers d'aide fournis avec Visual Basic Express sont un sous-ensemble limité de MSDN Library pour les éditions Visual Studio 2005 Express qui sont elles-mêmes un sous-ensemble de la bibliothèque MSDN Online Library intégrale. Lors de l'installation, vous avez pu choisir d'installer MSDN Library pour les éditions Visual Studio 2005 Express (si vous avez choisi de ne pas l'installer, il est conseillé de le faire maintenant).

De la documentation supplémentaire sur les produits et des versions mises à jour des rubriques fournies dans MSDN Library pour les éditions Visual Studio 2005 Express sont disponibles dans MSDN Online Library. Pour accéder à MSDN Online Library, choisissez **Centre pour développeurs** dans le menu **Communauté**, puis choisissez **Bibliothèque** sur la page d'accueil du Centre pour développeurs Microsoft Visual Basic.

Remarque

Pour accéder à MSDN Online Library, vous devez disposer d'une connexion active à Internet, et tout serveur proxy ou pare-fe u doit être configuré pour y autoriser l'accès.

Ressources en ligne

Étant donné que des millions de programmeurs utilisent Visual Basic dans le monde entier, il n'est pas surprenant qu'il existe de nombreuses ressources en ligne permettant d'en apprendre davantage sur Visual Basic. Une recherche MSN sur le mot clé Visual Basic retourne plus de 49 millions de résultats et une recherche sur le mot clé Visual Basic 2005 en retourne 2 millions. Avec un tel nombre de ressources en ligne, par où commencer ?

L'endroit le mieux adapté à la recherche de ressources d'apprentissage en ligne est le Centre pour développeurs Microsoft Visual Basic, disponible dans le menu **Communauté**. Le Centre pour développeurs est mis à jour en permanence avec de nouveaux articles sur Visual Basic, ainsi qu'avec des liens vers des manuels en ligne, des supports multimédia et des webdiffusions, des procédures pas à pas, et bien d'autres choses encore. Il vous suffit de cliquer sur le lien **Learning Visual Basic** de la page d'accueil du Centre pour développeurs Microsoft Visual Basic.

Vous pouvez également commencer votre recherche par le site CodeWise Community, également disponible dans le menu **Communauté**. Le site CodeWise Community est une ressource en ligne permettant d'obtenir des astuces de tiers, des exemples de code, des conseils et des informations émanant d'experts de Visual Basic et Visual Basic 2005.

Les membres de la Communauté CodeWise sont des experts en outils et technologies de développement Microsoft. Ils fournissent une vaste gamme de contenu par le biais de leurs communautés en ligne, et offrent des possibilités de partager vos opinions et de connaître celles d'autres utilisateurs.

Ressources supplémentaires

Si vous préférez des méthodes d'apprentissage plus traditionnelles, vous avez de nombreuses autres possibilités.

Il existe de nombreux manuels traitant de Visual Basic 2005, allant des manuels de niveau débutant à des manuels sur des utilisations spécifiques de Visual Basic 2005, tels que la programmation de jeux. Vous pouvez parcourir la section Programmation informatique de votre bibliothèque locale, librairie ou librairie en ligne pour savoir ce qui est actuellement disponible.

Conseil

Assurez-vous que les manuels sélectionnés traitent de Visual Basic 2005 et non d'une version antérieure de Visual Basic. Mê me si une grande partie des documents écrits pour les versions antérieures est toujours d'application, il arrive fréquemment que de nouvelles techniques plus performantes soient disponibles dans Visual Basic 2005.

De nombreux collèges, universités et écoles techniques proposent des cours traitant de Visual Basic 2005, certains visent même spécifiquement Visual Basic Express. Recherchez les cours disponibles dans votre région.

Étapes suivantes

Dans cette leçon, vous avez appris certaines façons de poursuivre l'apprentissage de Visual Basic. Dans la leçon suivante, vous apprendrez comment rechercher et utiliser un exemple de code.

Leçon suivante : Voler n'est pas un crime : exemples et Starter Kits.

Voir aussi

Tâches

Ressources communautaires : l'union fait la force

Concepts

Étape suivante : que faire ensuite ?

Comment faire dans Visual Basic Express

Autres ressources

Visite guidée de Visual Basic

Plus en profondeur : je souhaite en savoir davantage sur...

La Visite guidée de Visual Basic vous a présenté certaines opérations réalisables à l'aide de Visual Basic Express et de Visual Basic 2005, mais il ne s'agit que de la partie visible de l'iceberg. Si vous souhaitez approfondir la matière abordée dans la visite guidée, commencez par ici.

Remarque

Si vous utilisez Visual Basic Express, certains des liens d'aide fournis sur cette page risquent de ne pas être disponibles, selon les options que vous avez choisies pendant l'installation. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Pour en savoir plus sur le langage Visual Basic

Structure de programme et conventions de codage

Contient la documentation sur la structure de base et les conventions de code de Visual Basic, par exemple les conventions d'affectation de noms, les commentaires dans le code et les limites de Visual Basic.

Fonctionnalités du langage Visual Basic

Offre une vue d'ensemble des principaux composants de Visual Basic.

Éléments déclarés en Visual Basic

Aborde les variables, constantes, énumérations, structures, propriétés, méthodes, procédures, arguments de procédure, retours de fonction, événements, délégués, interfaces et classes.

Procédures dans Visual Basic

Décrit les procédures **Sub**, **Function**, **Property** et **Operator**, ainsi que des sujets plus avancés, tels que les procédures récurrentes et surchargées.

Flux de contrôle en Visual Basic

Décrit comment contrôler le déroulement de l'exécution d'un programme.

Développement avec My

Présente une nouvelle fonctionnalité appelée **My**, qui donne accès à des informations et des instances d'objet par défaut associées à une application et son environnement d'exécution.

Pour en savoir plus sur les Windows Forms et les contrôles

Mise en route des Windows Forms

Décrit en détail comment maîtriser la puissance des Windows Forms pour afficher des données, gérer les entrées d'utilisateur et déployer vos applications avec facilité et en toute sécurité.

Amélioration des applications Windows Forms

Décrit les fonctionnalités améliorées qui vous permettent de générer des applications Windows de classe internationale.

Applications Windows (Comment faire en Visual Basic)

Cette page fournit des liens vers l'aide sur les tâches Visual Basic largement utilisées pour créer des applications Windows.

Contrôles à utiliser dans les Windows Forms

Liste alphabétique des contrôles et composants pouvant être utilisés sur des Windows Forms.

Classement par fonction des contrôles Windows Forms

Répertorie les contrôles et les composants Windows Forms par fonction générale.

Nouveautés dans les Windows Forms et les contrôles Windows Forms

Énumère les fonctionnalités nouvelles et améliorées disponibles dans cette version de Visual Basic 2005.

Pour en savoir plus sur le débogage et la gestion des erreurs

Débogage de l'application Visual Basic

Cette page fournit des pointeurs vers la documentation relative aux fonctions de débogage intégrées dans Visual Basic 2005.

Comment : corriger des erreurs d'exécution avec l'Assistant Exception

L'**Assistant Exception** qui apparaît dans le débogueur à chaque exception d'exécution affiche le type d'exception, les conseils de dépannage et les remèdes qui peuvent être apportés par son intermédiaire.

Comment : appliquer des modifications en mode arrêt à l'aide de la fonctionnalité Modifier & Continuer

Vous pouvez utiliser Modifier & Continuer pour modifier votre code en mode arrêt, puis continuer sans arrêter et relancer l'exécution.

Comment : définir un point d'arrêt simple

Le débogueur de Visual Studio offre de nombreux moyens de définir des points d'arrêt. Cette rubrique présente deux façons rapides de définir des points d'arrêt simples.

Comment : corriger les erreurs de compilateur à l'aide de la correction automatique

Explique comment corriger les erreurs de compilation dans l'éditeur de code en utilisant la fonctionnalité **Correction automatique de la compilation intelligente**.

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Décrit et illustre la gestion structurée des exceptions dans Visual Basic.

Pour en savoir plus sur les bases de données

Accès aux données dans les applications Visual Basic

Visual Basic inclut plusieurs nouvelles fonctionnalités qui vous aident à développer des applications qui accèdent aux données.

Vue d'ensemble de l'affichage des données

Affichez des données aux utilisateurs de votre application en liant des données à des contrôles situés sur un formulaire. Créez des contrôles liés aux données en faisant glisser des éléments depuis la fenêtre **Sources de données** jusqu'à un Windows Form.

Vue d'ensemble des sources de données

Les sources de données représentent les données mises à la disposition de votre application et s'affichent dans la fenêtre **Sources de données**.

Vue d'ensemble de TableAdapter

Les **TableAdapters** assurent la communication entre votre application et une base de données.

Procédure pas à pas : affichage de données connexes sur un formulaire dans une application Windows

Fournit des détails pas à pas sur l'affichage de données provenant de deux tables connexes et leur affichage sur un Windows Form.

Validation des données

Les groupes de données fournissent des événements spécifiques qui sont déclenchés lors de la modification de valeurs dans des lignes de données. Vous pouvez créer des gestionnaires d'événements pour valider ces modifications en fonction des exigences de votre application.

Pour en savoir plus sur la programmation orientée objet

Introduction aux objets dans Visual Basic

Fournit une introduction aux termes et aux concepts utilisés dans la programmation orientée objet.

Création et utilisation d'objets

Montre comment créer et utiliser des instances de classes.

Gestion de groupes d'objets

Fournit des techniques de travail sur les tableaux et les collections d'objets

Présentation des classes

Présente des instructions détaillées sur la création d'objet et les problèmes de durée de vie.

Interfaces en Visual Basic

Définit les interfaces et présente leur utilisation dans vos applications

Définit les interfaces et présente leur utilisation dans vos applications.

Héritage en Visual Basic

Explique comment définir des classes servant de base à d'autres classes.

Pour en savoir plus sur les contrôles utilisateur et les composants

Property, instruction

Déclare le nom d'une propriété, ainsi que les procédures de propriété utilisées pour stocker et récupérer la valeur de la propriété.

Création et utilisation de composants en Visual Basic

Définit le terme « composant » et explique quand et comment créer des composants.

Comment : utiliser les contrôles ActiveX

Montre comment utiliser des contrôles ActiveX existants pour ajouter des fonctionnalités à la **Boîte à outils** de Visual Studio.

Constantes et énumérations en Visual Basic

Décrit le stockage de valeurs inchangées pour un usage fréquent.

Vue d'ensemble des énumérations

Décrit le stockage d'ensembles de valeurs de constantes associées et l'attribution de noms à ces ensembles.

Pour en savoir plus sur les graphiques

Graphiques et dessins dans les Windows Forms

Contient des liens vers des rubriques qui décrivent et montrent comment utiliser l'interface graphique dans les Windows Forms.

Mise en route de la programmation graphique

Décrit comment exécuter des tâches de base avec GDI+.

Utilisation de polices et de texte

Montre comment dessiner du texte et utiliser des polices et des familles de polices.

Utilisation de conteneurs graphiques

Montre comment gérer les états des objets graphiques et les conteneurs graphiques imbriqués.

Graphiques mis deux fois en mémoire tampon

Décrit comment réduire le scintillement à l'aide du mécanisme de double tampon.

Contrôles avec prise en charge intégrée des dessins owner-drawn

Le dessin owner-drawn dans Windows Forms, qui est également qualifié de dessin personnalisé, est une technique qui consiste à modifier l'apparence visuelle de certains contrôles.

Pour en savoir plus sur le déploiement

Vue d'ensemble du déploiement ClickOnce

Décrit la fonction de ClickOnce et explique comment il résout des problèmes importants liés au déploiement des applications Windows Forms sur les ordinateurs clients.

Choix d'une stratégie de déploiement ClickOnce

Décrit les différentes manières de déployer une application ClickOnce sur des ordinateurs clients, en spécifiant les avantages et les limitations de chaque stratégie.

Choix d'une stratégie de mise à jour ClickOnce

Décrit les différentes options disponibles pour mettre à jour d'une application ClickOnce après son installation, et notamment les abonnements de mise à jour automatisés et les mises à jour à la demande.

Dépannage des déploiements ClickOnce

Décrit les problèmes fréquemment rencontrés lors du déploiement d'applications ClickOnce et explique comment les résoudre.

Voir aussi

Tâches

Voler n'est pas un crime : exemples et Starter Kits

Concepts

Étape suivante : que faire ensuite ?

Comment faire dans Visual Basic Express

Autres ressources

Visite guidée de Visual Basic

Réutilisation de code existant

Écrire du code, c'est bien, mais ne pas devoir en écrire, c'est mieux encore ! Visual Basic 2005 facilite la réutilisation du code contenu dans les rubriques d'aide, les exemples d'applications et les Starter Kits.

Remarque

Si vous utilisez Visual Basic Express, certains des liens d'aide fournis sur cette page risquent de ne pas être disponibles, selon les options que vous avez choisies pendant l'installation. Pour plus d'informations, consultez Dépannage de Visual Basic Express.

Copie de code

Au fil des leçons, vous avez pu constater qu'à côté des blocs de code fournis dans les rubriques d'aide se trouvaient également des liens marqués **Copier le code**. Si vous cliquez sur le lien **Copier le code** d'une rubrique d'aide, le code contenu dans ce bloc est copié dans le Presse-papiers ; vous pouvez ensuite le coller directement dans l'éditeur de code plutôt que de le taper.

De nombreuses rubriques d'aide contiennent des exemples de code montrant la manière dont vous pouvez utiliser un élément, une propriété ou une fonction particuliers du langage. Même s'ils ne correspondent pas exactement aux opérations que vous souhaitez exécuter, vous pouvez les copier dans votre code comme point de départ, puis les modifiez en fonction de vos besoins en changeant les noms des variables et les références.

Utilisation des exemples d'applications

Visual Basic 2005 comprend une série d'*exemples d'applications* qui peuvent vous donner une longueur d'avance pour la création de vos propres programmes. Les exemples d'applications sont des projets complets qui peuvent être chargés dans Visual Basic 2005 et être exécutés sans apporter de modification. Dans la plupart des cas, ils ne sont pas très utiles comme programmes ; toutefois, ils contiennent souvent une grande quantité de code utile.

Par exemple, l'exemple d'application Game illustre un jeu Windows Forms unique complet avec des graphiques GDI+, des fonctionnalités de minuterie, une configuration utilisateur et un stockage des meilleurs scores. Si vous écrivez votre propre jeu de cartes, le code de l'interface utilisateur fourni dans l'exemple Game ne vous sera probablement pas d'une grande utilité. Par contre, le code permettant de stocker et d'afficher les meilleurs scores vous sera très utile ; vous pourrez le copier et l'utiliser tel quel dans votre programme.

Dans d'autres cas, vous pouvez prendre l'exemple d'application et l'utiliser comme point de départ, en le modifiant et en ajoutant du code en fonction de vos propres besoins. Même si l'exemple n'exécute pas exactement l'opération souhaitée, cela est souvent plus facile que de commencer à zéro. Pour plus d'informations, consultez Exemples d'applications Visual Basic Express.

Starter Kits

Les Starter Kits sont semblables aux exemples d'applications, mais contrairement à la plupart des exemples, il s'agit d'applications complètes qui peuvent être utilisées de manière autonome. Par exemple, la Starter Kit Ma cinémathèque est une application complète assurant le suivi de votre cinémathèque de DVD.

Comme les exemples d'applications, les Starter Kits peuvent être utilisés comme point de départ pour votre propre programme. Si vous souhaitez créer un programme assurant le suivi des fichiers musicaux plutôt que des films, vous pouvez aisément personnaliser Ma cinémathèque en fonction de vos besoins. Les Starter kits contiennent des suggestions pour la personnalisation, avec l'exemple de code, vous aider. Pour plus d'informations, consultez Starter Kits : pour être rapidement opérationnel.

Réutilisation de votre propre code

Lorsque vous aurez programmé pendant un certain temps, vous constaterez probablement que vous écrivez le même code à de nombreuses reprises. Par exemple, vous pouvez écrire du code pour un contrôle **TextBox** afin de n'autoriser que la saisie de chiffres. Plutôt que d'écrire le même code pour chaque programme, vous pouvez écrire le code à une reprise et l'enregistrer en le faisant glisser jusqu'à la **Boîte à outils**. La prochaine fois que vous avez besoin de cet extrait de code, vous pouvez le faire glisser à nouveau jusqu'à l'éditeur de code pour l'insérer dans votre code.

Conseil

Si vous enregistrez une grande quantité de code dans la **Boîte à outils**, il peut être compliqué de trouver le code dont vous a vez besoin. Vous pouvez également l'enregistrer en tant qu'extrait de code et y accéder exactement de la même manière qu'a ux extraits de code fournis avec Visual Basic 2005.

Recherche de code

Outre le code fourni dans Visual Basic Express, il existe de nombreuses autres sources permettant d'obtenir du code (de Microsoft et d'autres fabricants). Vous pouvez rechercher des extraits de code, des modèles et des Starter kits, des exemples ainsi que des contrôles en choisissant **Rechercher dans les communautés** dans le menu **Communauté**. Pour plus d'informations, consultez Comment : rechercher des extraits de code en ligne.

Vous pouvez également parcourir l'aide pour rechercher des rubriques qui contiennent du code. Pour plus d'informations, consultez Comment : rechercher des rubriques avec des exemples de code.

Étapes suivantes

Dans cette leçon, vous avez appris les différentes options permettant de rechercher du code afin de ne pas avoir à l'écrire vous-même. Dans la leçon suivante, vous saurez où obtenir des informations détaillées sur les différentes fonctionnalités de Visual Basic Express.

Leçon suivante : Plus en profondeur : je souhaite en savoir davantage sur....

Voir aussi

Tâches

Informations complémentaires : ressources pour l'apprentissage de Visual Basic

Concepts

Étape suivante : que faire ensuite ?

Comment faire dans Visual Basic Express

Autres ressources

Visite guidée de Visual Basic

Aide pour les utilisateurs de Visual Basic 6.0

En tant que développeur chevronné, vous avez consacré beaucoup de temps et d'efforts à Visual Basic 6.0. À première vue, Visual Basic 2005 peut sembler radicalement différent et donner l'impression qu'il vous faut tout réapprendre. Toutefois, malgré les nombreuses modifications apportées, le principe général de développement en vigueur dans Visual Basic 6.0 demeure inchangé. Votre connaissance de Visual Basic 6.0 vous aidera rapidement à devenir productif avec Visual Basic 2005.

Vous devrez vous familiariser avec un certain nombre de modifications apportées à Visual Basic 6.0, et mettre de côté certaines habitudes antérieures. Certaines règles liées au langage, certaines tâches, notamment le débogage et le déploiement, et même certains termes, ont changé, souvent de façon subtile. Enfin, les modifications rendent Visual Basic 2005 plus facile et plus puissant.

Les rubriques suivantes vous aideront à identifier les modifications introduites dans Visual Basic 2005.

Dans cette section

Environnement de développement intégré pour les utilisateurs de Visual Basic 6.0

Modifications à l'environnement de développement.

Gestion de projets pour les utilisateurs de Visual Basic 6.0

Modifications au système de projet.

Programmation Web pour les utilisateurs de Visual Basic 6.0

Modifications au développement Web.

WebClasses pour les utilisateurs de Visual Basic 6.0

Modifications aux projets WebClass (également appelés projets d'application IIS).

Accès aux données pour les utilisateurs de Visual Basic 6.0

Modifications aux outils de données et d'accès aux données.

Création de composants pour les utilisateurs de Visual Basic 6.0

Modifications à la création des composants.

Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0

Modifications aux projets **UserControl** (également connus sous le nom de projets de contrôle ActiveX). Fournit un lien vers des informations sur l'objet **PropertyBag**.

Windows Forms pour les utilisateurs de Visual Basic 6.0

Modifications aux formulaires.

Débogage pour les utilisateurs de Visual Basic 6.0

Modifications au débogage.

Installation et déploiement pour les utilisateurs de Visual Basic 6.0

Modifications à l'installation et au déploiement.

Localisation et globalisation pour les utilisateurs de Visual Basic 6.0

Modifications à la localisation.

Programmation de l'API Windows pour les utilisateurs de Visual Basic 6.0

Modifications à l'accès à l'API Windows.

Accès au Registre pour les utilisateurs de Visual Basic 6.0

Modifications à la définition et à la récupération des paramètres du Registre.

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Modifications aux constantes qui ont changé dans Visual Basic 2005.

Bibliothèques d'objets et espaces de noms pour les utilisateurs de Visual Basic 6.0

Différences et ressemblances entre les bibliothèques d'objets Visual Basic 6.0 et les espaces de noms de Visual Basic 2005.

Bibliothèques Runtime pour les utilisateurs de Visual Basic 6.0

Différences entre les fichiers exécutables Visual Basic 6.0 et le Common Language Runtime du .NET Framework.

Rubriques connexes

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Modifications au langage Visual Basic.

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Modifications aux contrôles.

Mise à niveau de Visual Basic 6.0

Conversion d'applications au format Visual Basic 2005.

Nouveautés de Visual Basic

Nouvelles fonctionnalités de Visual Basic 2005.

Nouveautés pour les utilisateurs de Visual Basic 6.0

Si vous êtes un utilisateur expérimenté de Visual Basic 6.0, vous trouverez dans Visual Basic 2005 de nombreuses fonctionnalités nouvelles ou considérablement améliorées, qui rendront le développement avec Visual Basic plus facile et plus puissant que dans les versions antérieures. La liste suivante met en évidence quelques-unes des modifications les plus importantes et fournit des liens vers les informations détaillées.

Nouveautés du langage Visual Basic

Visual Basic comporte de nombreuses fonctionnalités de langage nouvelles et améliorées, telles que l'héritage, les interfaces et la surcharge, qui en font un langage de programmation orienté objet puissant. En tant que développeur Visual Basic, vous pouvez maintenant créer de véritables applications multithread. La gestion structurée des exceptions, les attributs personnalisés et la conformité CLS (Common Language Specification) comptent parmi les autres nouvelles fonctionnalités de langage introduites par Visual Basic 2005.

Visual Basic 2005 prend en charge de nombreuses fonctionnalités de langage orientées objet, nouvelles ou améliorées, telles que l'héritage, la surcharge, le mot clé **Overrides**, les interfaces, les membres partagés et les constructeurs.

Pour plus d'informations sur les nouveautés du langage Visual Basic, consultez Nouveautés dans le langage Visual Basic pour les utilisateurs de Visual Basic 6.0.

Pour obtenir un examen des fonctions et fonctionnalités ayant fait l'objet de modifications depuis les versions antérieures, consultez Modifications du langage pour les utilisateurs de Visual Basic 6.0.

Développement Visual Basic avec My

Visual Basic 2005 fournit de nouvelles fonctionnalités destinées au développement rapide d'application. Celles-ci ont pour objectif d'améliorer la productivité et de faciliter l'exploitation, tout en fournissant de la puissance. L'une de ces fonctionnalités, nommée **My**, fournit un accès aux fonctionnalités les plus couramment utilisées du .NET Framework, ainsi que des informations et des instances d'objet par défaut associées à l'application et à son environnement d'exécution. Ces informations sont organisées dans un format qui autorise une navigation IntelliSense et dont la logique s'organise en fonction de l'utilisation. Pour plus d'informations, consultez Développement avec My.

Nouveautés dans les formulaires et les contrôles

Windows Forms est la nouvelle structure orientée objet qui implémente le .NET Framework. Windows Forms, en association avec les contrôles Windows Forms, fournissent une architecture fiable pour le développement d'applications Windows en Visual Basic.

Pour plus d'informations sur les nouveautés introduites dans les formulaires et les contrôles, consultez Nouveautés dans les Windows Forms pour les utilisateurs de Visual Basic 6.0.

Si les formulaires de Visual Basic 6.0 vous sont familiers, consultez Tâches des formulaires pour les utilisateurs de Visual Basic 6.0 pour une introduction aux nouveaux concepts.

Nouveautés de l'IDE Visual Basic

À première vue, l'environnement de développement intégré Visual Basic 2005 peut paraître peu familier, mais vous ne tarderez pas à découvrir, à mesure que vous l'utiliserez, qu'il offre de nombreuses fonctionnalités nouvelles qui peuvent vous rendre plus productif.

Pour plus d'informations, consultez Nouveautés dans l'environnement IDE de Visual Studio pour les utilisateurs de Visual Basic 6.0.

Nouveautés des données

Visual Basic 2005 inclut plusieurs nouvelles fonctionnalités qui vous aident à développer des applications qui accèdent aux données. L'Assistant Configuration de source de données simplifie la connexion de votre application à des données présentes dans des bases de données, des services Web et des objets créés par utilisateur. Le nouveau Sources de données, fenêtre fournit un emplacement central pour consulter les données disponibles et associées à votre projet. Il réduit la complexité de la liaison de données en autorisant le glissement d'éléments de la fenêtre vers des formulaires en vue de créer des contrôles liés aux données. Le remplissage des groupes de données, l'exécution de requêtes et l'exécution de procédures stockées peuvent désormais s'effectuer à l'aide du nouvel objet Vue d'ensemble de TableAdapter généré par Visual Studio. La nouvelle fonctionnalité de données locale vous permet d'intégrer des fichiers de base de données Microsoft Access et Microsoft SQL Server Express directement à votre application. Pour une liste complète des nouvelles fonctionnalités, consultez

Nouveautés des données.

Nouveautés de l'installation et du déploiement

Le déploiement d'applications créées dans Visual Basic 2005 est plus facile et plus puissant que jamais, grâce à de nouvelles technologies telles que le déploiement ClickOnce.

Le déploiement ClickOnce vous permet de publier des applications Windows et console à mise à jour automatique, qui peuvent être installées, mises à jour, et exécutées aussi facilement que des applications Web. Les nouveaux onglets **Sécurité**, **Signature** et **Publier** du **Concepteur de projets** permettent de personnaliser votre déploiement ClickOnce. La nouvelle commande **Publier** du menu **Générer** (et du menu contextuel de l'**Explorateur de solutions**) permet d'accéder à l'**Assistant Publication**, un outil qui vous guide étape par étape dans la publication de votre application. Pour plus d'informations, consultez Déploiement ClickOnce.

De plus, l'Assistant **Configuration et déploiement** de Visual Basic 6.0 est remplacé dans Visual Basic 2005 par les projets de configuration et de déploiement fondés sur Windows Installer, le programme d'installation de Windows. Pour plus d'informations, consultez Installation et déploiement pour les utilisateurs de Visual Basic 6.0

Mise à niveau des applications créées dans Visual Basic 6.0

Visual Studio 2005 vous donne la possibilité de mettre à niveau des applications créées avec Visual Basic 6.0 et ainsi de continuer leur développement en bénéficiant des avantages du .NET Framework. Lorsque vous ouvrez pour la première fois un fichier projet Visual Basic 6.0 (.vbp), l'Assistant Mise à niveau apparaît. Un outil de ligne de commande est également fourni pour la mise à niveau de projets en dehors de l'environnement de développement. Pour plus d'informations, consultez Mise à niveau d'applications créées dans des versions précédentes de Visual Basic.

Voir aussi

Concepts

Nouveautés de Visual Basic

Autres ressources

Visite guidée de Visual Basic

Nouveautés dans le langage Visual Basic pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 comporte de nombreuses fonctionnalités de langage nouvelles et améliorées, telles que l'héritage, les interfaces et la surcharge, qui en font un langage de programmation orienté objet puissant. Cette page met en évidence quelques-unes des plus importantes modifications et fournit des liens vers les informations détaillées.

Remarque

Si Visual Basic 6.0 vous est familier, consultez Modifications du langage pour les utilisateurs de Visual Basic 6.0 pour une introduction aux nouveaux concepts.

Conseil

Pour une introduction pratique aux nouvelles fonctionnalités de langage dans Visual Basic 2005, consultez Visite guidée de Visual Basic.

Nouveautés

Les nouvelles fonctionnalités du langage Visual Basic 2005 sont les suivantes :

Héritage

Visual Basic 2005 prend en charge l'*héritage* en vous permettant de définir les classes qui constitueront la base de classes dérivées. Les classes dérivées héritent des propriétés et des méthodes de la classe de base, et peuvent les étendre. Elles peuvent également substituer des méthodes héritées avec de nouvelles implémentations de la classe de base. Toutes les classes créées avec Visual Basic 2005 peuvent être héritées par défaut. Les formulaires que vous concevez constituent de véritables classes. Aussi, vous pouvez utiliser l'héritage pour définir de nouveaux formulaires qui reposent sur des formulaires existants. Pour plus d'informations, consultez Héritage en Visual Basic.

Gestion des exceptions

Visual Basic 2005 prend en charge la *gestion structurée des exceptions*, à l'aide d'une version améliorée de la syntaxe **Try...Catch...Finally** prise en charge par d'autres langages, tels que C++. La gestion structurée des exceptions associe une structure de contrôle moderne (semblable à **Select Case** ou **While**) à des exceptions, des blocs de code protégés et des filtres. La gestion structurée des exceptions facilite la création et la maintenance des programmes au moyen de gestionnaires d'erreurs robustes et exhaustifs. Pour plus d'informations, consultez Introduction à la gestion des exceptions.

Surcharge

La *surcharge* est la capacité de définir des propriétés, méthodes, procédures ou opérateurs qui portent le même nom, mais utilisent des types de données différents. Des procédures surchargées vous permettent de fournir autant d'implémentations que nécessaire pour gérer différents types de données, tout en donnant l'impression d'une procédure unique et flexible. Pour plus d'informations, consultez Propriétés et méthodes surchargées.

Substitution de propriétés et de méthodes

Le mot clé **Overrides** permet aux objets dérivés de substituer les caractéristiques héritées des objets parents. Les membres substitués ont les mêmes arguments que les membres hérités de la classe de base, mais ils ont des implémentations différentes. La nouvelle implémentation d'un membre peut appeler l'implémentation originale dans la classe parente, en faisant précéder le nom du membre de **MyBase**. Pour plus d'informations, consultez Substitution de propriétés et de méthodes.

Constructeurs et destructeurs

Les *constructeurs* sont des procédures qui contrôlent l'initialisation de nouvelles instances d'une classe. Inversement, les *destructeurs* sont des méthodes qui libèrent des ressources système lorsqu'une classe abandonne la portée ou se voit attribuer la valeur **Nothing**. Visual Basic 2005 prend en charge les constructeurs et les destructeurs à l'aide des procédures **Sub New** et **Sub Finalize**. Pour plus d'informations, consultez Durée de vie d'un objet : création et destruction des objets.

Types de données

Visual Basic 2005 introduit trois nouveaux types de données. Le type de données **Char** est une quantité 16 bits non signée utilisée pour stocker des caractères Unicode. Il est équivalent au type de données .NET Framework **System.Char**. Le type de

données **Short**, un nombre entier 16 bits signé, était nommé **Integer** dans les versions antérieures de Visual Basic. Le type de données **Decimal** est un entier 96 bits signé mis à l'échelle par une puissance de 10 variable. Dans les versions antérieures de Visual Basic, il était uniquement disponible dans un **Variant**. De plus, Visual Basic prend maintenant en charge des types de données d'entier non signé (**UShort**, **UInteger** et **ULong**), ainsi que type signé **SByte**. Pour plus d'informations, consultez Types de données en Visual Basic.

Interfaces

Les *Interfaces* décrivent les propriétés et les méthodes des classes, mais à la différence des classes, elles ne fournissent pas les implémentations. Utilisez l'instruction **Interface** pour déclarer des interfaces, et l'instruction **Implements** pour écrire du code qui mette en application les éléments décrits dans l'interface. Pour plus d'informations, consultez Interfaces en Visual Basic.

Délégués

Les *délégués*, objets qui peuvent appeler les méthodes des objets pour votre compte, sont parfois décrits comme des pointeurs fonction orientés objet de type sécurisé. Vous pouvez utiliser des délégués pour laisser les procédures indiquer une méthode de gestionnaire d'événements qui s'exécute lorsqu'un événement se produit. Vous pouvez également utiliser des délégués avec les applications multithread. Pour plus d'informations, consultez Délégués et opérateur AddressOf.

Membres partagés

Les *membres partagés* sont des propriétés, des procédures et des champs qui sont partagés par toutes les instances d'une classe. Les données membres partagées sont utiles lorsque plusieurs objets doivent utiliser des informations communes à tous. Vous pouvez utiliser des méthodes de classe partagées sans créer au préalable d'objet issu d'une classe. Pour plus d'informations, consultez Membres partagés en Visual Basic.

Références

Les *références* vous permettent d'utiliser des objets définis dans d'autres assemblys. Dans Visual Basic 2005, les références pointent sur des assemblys plutôt que sur des bibliothèques de types. Pour plus d'informations, consultez Références et l'instruction Imports.

Espaces de noms

Les *espaces de noms* empêchent les conflits de dénomination en organisant les classes, les interfaces et les méthodes en hiérarchies. Pour plus d'informations, consultez Espaces de noms dans Visual Basic.

Assemblys

Les *assemblys* remplacent et étendent les capacités des bibliothèques de types en décrivant tous les fichiers requis pour un composant ou application donnée. Chaque assembly peut comporter un ou plusieurs espaces de noms. Pour plus d'informations, consultez Assemblys.

Attributs

Vous pouvez utiliser des *attributs* pour fournir des informations supplémentaires relatives aux éléments d'un programme. Par exemple, vous pouvez utiliser un attribut pour indiquer les méthodes d'une classe qui doivent être exposées lorsque cette classe est employée en tant que service Web XML. Pour plus d'informations, consultez Attributs en Visual Basic.

Multithreading

Vous pouvez utiliser Visual Basic 2005 pour écrire des applications qui peuvent exécuter indépendamment plusieurs tâches. Une tâche qui peut suspendre d'autres tâches peut s'exécuter sur un thread distinct ; ce processus est connu sous le nom de *multithreading*. En exécutant des tâches complexes sur des files de traitement, ou threads, séparées de votre interface utilisateur, le multithreading rend vos applications plus réactives aux entrées d'utilisateur. Pour plus d'informations, consultez Applications multithread.

Opérateurs de décalage de bits

Visual Basic 2005 prend désormais en charge les opérations de décalage arithmétiques vers la gauche et la droite sur les types de données intégraux (**Byte**, **Short**, **Integer** et **Long**), ainsi que sur les types non signés (**UShort**, **UInteger** et **ULong**). Les décalages arithmétiques ne sont pas circulaires ; ce qui signifie que les bits décalés à une extrémité du résultat ne sont pas réintroduits à l'autre. Les opérateurs d'assignation correspondants sont également fournis. Pour plus d'informations, consultez Opérateurs de décalage de bits et Opérateurs d'assignation.

Déclaration de variables de boucle

Vous pouvez utiliser Visual Basic 2005 pour déclarer une variable de boucle dans le cadre d'une boucle **For** ou **For Each**. Vous pouvez inclure une clause **As** pour la variable dans l'instruction **For** ou **For Each** si aucune variable de même nom n'a été déclarée hors de la boucle. La portée d'une variable de boucle déclarée de cette manière correspond à la boucle elle-même. Pour plus d'informations, consultez For...Next, instruction (Visual Basic) et For Each...Next, instruction (Visual Basic).

Nouveautés du langage Visual Basic dans Visual Basic 2005

Dans cette version de Visual Studio, les nouvelles fonctionnalités de langage introduites dans Visual Basic 2005 comptent notamment la continuation de boucle, la mise à disposition de ressources garanties, les propriétés d'accès mixte, les types de données non signés, la surcharge d'opérateur ou les types génériques. Pour plus d'informations, consultez Nouveautés du langage Visual Basic.

Voir aussi

Concepts

Nouveautés pour les utilisateurs de Visual Basic 6.0
Nouveautés dans les Windows Forms pour les utilisateurs de Visual Basic 6.0
Nouveautés dans l'environnement IDE de Visual Studio pour les utilisateurs de Visual Basic 6.0

Nouveautés dans les Windows Forms pour les utilisateurs de Visual Basic 6.0

Windows Forms est la nouvelle structure orientée objet qui implémente le .NET Framework. Windows Forms et les contrôles Windows Forms fournissent une architecture fiable pour le développement d'applications Windows en Visual Basic.

Remarque

Si les formulaires de Visual Basic 6.0 vous sont familiers, consultez
Tâches des formulaires pour les utilisateurs de Visual Basic 6.0 pour une introduction aux nouveaux concepts.

Nouveautés

Ces nouveautés concernent toutes le développement d'applications clientes à l'aide de Windows Forms.

Windows Forms

Windows Forms est la nouvelle plate-forme basée sur le .NET Framework, pour le développement d'applications Microsoft Windows. Vous pouvez utiliser cette infrastructure pour créer des applications Windows riches et puissantes à l'aide d'un jeu de classes orienté objet, clair et extensible. Pour plus d'informations, consultez Vue d'ensemble des Windows Forms.

Contrôles Windows Forms

Visual Basic 2005 intègre tous les contrôles Visual Basic 6.0 qui vous sont familiers ; la plupart des contrôles sont améliorés au moyen de nouveaux événements, propriétés et méthodes qui les rendent plus puissants. De plus, de nouveaux contrôles et composants permettent de créer votre interface utilisateur. Pour plus d'informations, consultez
Contrôles à utiliser dans les Windows Forms.

Windows Forms comme niveau de présentation d'une application à trois couches

Au moyen d'appels HTTP à un serveur connecté à une base de données, les Windows Forms peuvent agir comme une interface utilisateur qui combine l'interface utilisateur d'une application cliente et le traitement avancé d'un serveur d'applications. Pour plus d'informations, consultez Procédure pas à pas : création d'une application distribuée.

Création d'une application cliente

Les Windows Forms permettent de créer une application cliente élaborée à l'aide de Visual Basic. Cette application est capable d'accéder à un large éventail de sources de données et propose des fonctionnalités d'affichage et de modification des données au moyen de contrôles Windows Forms. Pour plus d'informations, consultez
Procédure pas à pas : création d'un Windows Form simple.

Modèle de sécurité des Windows Forms

La sécurité des Windows Forms repose sur les stratégies de sécurité établies au sein du .NET Framework. L'espace de noms System.Security fournit la structure sous-jacente du système de sécurité du .NET Framework, notamment les classes de base des autorisations. Pour plus d'informations, consultez Sécurité des Windows Forms.

Propriétés Tag et Name

Les propriétés **Tag** et **Name** font désormais partie du contrôle. La propriété **Tag** est un référentiel en mémoire utilisé pour stocker des données sur le contrôle. La propriété **Name** contient le nom du contrôle. La propriété **Name** peut être utilisée au moment de l'exécution pour évaluer l'objet par nom, plutôt que par type et nom programmé. Pour plus d'informations sur la propriété **Tag**, consultez Control.Tag Property. Pour plus d'informations sur la propriété **Name**, consultez Control.Name Property.

Windows Forms non rectangulaires

Vous pouvez personnaliser facilement la forme des Windows Forms. Pour les formulaires non rectangles, vous avez simplement besoin d'un fichier image et de quelques paramètres de propriété. Pour plus d'informations, consultez
Comment : créer un Windows Form mis en forme.

Programmabilité des Smart Device

Vous pouvez maintenant utiliser Visual Basic pour développer des applications riches pour les périphériques soumis à des contraintes de ressources. Vous pouvez utiliser les outils fournis par Visual Studio 2005 et le .NET Compact Framework, un sous-ensemble du .NET Framework, pour créer, générer et déboguer des applications qui s'exécutent sur des périphériques

intelligents.

Le langage Visual Basic est presque entièrement pris en charge en termes de développement de périphériques. Toutefois, certaines fonctionnalités de Visual Basic 2005 ont été modifiées, voire supprimées, à des fins d'adaptation aux différences inhérentes aux applications de périphériques. Pour plus d'informations, consultez
Différences de développement bureautique dans le développement .NET Compact Framework.

Nouveautés des Windows Forms et des contrôles pour Visual Studio 2005

En plus des fonctionnalités répertoriées ci-dessus, plusieurs fonctionnalités et améliorations sont totalement nouvelles dans Visual Basic 2005, notamment plusieurs contrôles, une prise en charge des opérations d'arrière-plan, et la fonction de dessin personnalisée de plusieurs contrôles. Pour plus d'informations, consultez
Nouveautés dans les Windows Forms et les contrôles Windows Forms.

Voir aussi

Concepts

Nouveautés pour les utilisateurs de Visual Basic 6.0
Nouveautés dans le langage Visual Basic pour les utilisateurs de Visual Basic 6.0
Nouveautés dans l'environnement IDE de Visual Studio pour les utilisateurs de Visual Basic 6.0

Nouveautés dans l'environnement IDE de Visual Studio pour les utilisateurs de Visual Basic 6.0

À première vue, l'environnement de développement intégré Visual Basic 2005 peut paraître peu familier, mais vous ne tarderez pas à découvrir, à mesure que vous l'utiliserez, qu'il offre de nombreuses fonctionnalités nouvelles qui peuvent vous rendre plus productif. Cette page met en évidence quelques-unes des plus importantes modifications et fournit des liens vers les informations détaillées.

Remarque

Si vous êtes familiarisé avec Visual Basic 6.0, consultez
Environnement de développement intégré pour les utilisateurs de Visual Basic 6.0.

Conseil

Pour une introduction pratique aux nouvelles fonctionnalités de Visual Basic 2005, consultez Visite guidée de Visual Basic.

Nouveautés

Ci-dessous certaines des nouvelles fonctionnalités de l'IDE de Visual Basic 2005.

IDE partagé

Tous les langages .NET, y compris Visual Basic, Visual C#, Visual C++ et d'autres, sont hébergés dans l'IDE de Visual Studio. Le fait de partager un environnement de développement intégré unique offre de nombreux avantages, et en particulier le regroupement d'outils semblables provenant des divers produits en un ensemble d'outils partagés utilisés partout dans Visual Studio.

Gestion des fenêtres

Avec la présente version de Visual Studio, il n'a jamais été aussi facile d'afficher encore plus de code sur un même écran.

Les documents avec onglet tabulent automatiquement les fenêtres de document ensemble dans l'IDE. Par exemple, lorsque vous travaillez sur plusieurs documents à la fois dans un éditeur ou un concepteur, ces documents apparaissent tous sous forme d'onglets en haut de la zone d'interface multidocument (MDI, Multiple-Document Interface).

Masquer automatiquement vous permet de réduire des fenêtres d'outil, telles que l'**Explorateur de solutions** et la **boîte à outils**, en bordure de l'environnement de développement intégré (IDE), de sorte que ces fenêtres n'occupent pas un espace précieux. En réduisant les fenêtres d'outil, vous pouvez augmenter l'espace à la disposition de l'éditeur sur l'écran.

Pour plus d'informations, consultez Gestion des fenêtres.

Modification des fonctionnalités

Visual Studio possède désormais pour tous les langages de l'environnement de développement intégré (IDE) un éditeur de code unifié qui contient des fonctionnalités spécialisées pour chaque langage. Cet éditeur de code bénéficie de plusieurs améliorations, telles que le retour automatique à la ligne, la recherche incrémentielle, le contour de code, la réduction aux définitions, la numérotation des lignes, l'impression des couleurs et les raccourcis. Vous pouvez accéder à ces fonctionnalités à partir du menu **Edition** ou des menus contextuels. Pour plus d'informations, consultez Nouveautés dans l'édition de code.

Les extraits de code sont des segments d'exemple de code prêts à être insérés dans des projets Visual Basic. Pour afficher une liste d'extraits de code disponibles, cliquez avec le bouton droit sur le document actif dans l'éditeur de code, puis cliquez sur **Insérer un extrait** dans le menu contextuel. Cliquez sur le nom de l'extrait de code voulu. Le code est ensuite inséré dans l'éditeur, prêt à être modifié à votre convenance. Pour plus d'informations, consultez Comment : gérer des extraits de code.

À l'instar des balises actives Office, les balises actives Visual Studio rendent disponibles les tâches courantes qui s'appliquent au contexte de votre travail. Par exemple, vous pouvez désormais corriger certaines erreurs courantes en Visual Basic d'un clic de bouton à l'aide de balises actives.

Déploiement

Le déploiement ClickOnce vous permet de déployer des applications Windows à mise à jour automatique pouvant être installées et exécutées aussi facilement que des applications Web. Vous pouvez déployer des applications clientes Windows et en ligne de commande. Il existe une nouvelle commande **Publier Projet** dans les menus **Projet**. Pour plus d'informations,

consultez Déploiement ClickOnce.

Vous pouvez maintenant inclure des composants systèmes requis, tels que l'exécutable .NET Framework, dans le cadre d'un projet de déploiement ou d'un déploiement ClickOnce. Pour plus d'informations, consultez Déploiement des composants requis (Visual Studio).

À l'aide de projets **Installation et déploiement**, vous pouvez distribuer des applications à l'aide de la technologie Microsoft Windows Installer, les déployez sur des serveurs de production et des serveurs intermédiaires, déployez des couches de votre application sur différents ordinateurs de test, et déployez des applications Web ASP.NET sur des serveurs Web. Pour plus d'informations, consultez Déploiement de Windows Installer.

Microsoft Build Engine

Le Microsoft Build Engine (MSBuild) est la nouvelle plate-forme de génération pour Microsoft et Visual Studio. MSBuild introduit un nouveau format de fichier projet XML, simple à comprendre, facile à étendre et intégralement pris en charge par Microsoft. Le format du fichier projet MSBuild permet aux développeurs de décrire intégralement les éléments à générer, aussi bien que la façon dont ils doivent l'être en fonction des différentes plates-formes et configurations. De plus, le format de fichier projet permet aux développeurs de créer des règles réutilisables qui peuvent être réparties en fichiers distincts, de telle sorte que les générations puissent être faites de façon cohérente sur différents projets du produit. Pour plus d'informations, consultez MSBuild.

Nouveautés de l'IDE de Visual Studio dans Visual Basic 2005

Dans cette version de Visual Studio, les nouvelles fonctionnalités de l'IDE dans Visual Basic 2005 incluent des paramètres prédéfinis, des améliorations de la liste des tâches et de la liste d'erreurs, un meilleur comportement d'ancrage, la fenêtre Navigateur IDE, et bien d'autres. Pour plus d'informations, consultez Nouveautés de l'environnement de développement.

Voir aussi

Concepts

Nouveautés pour les utilisateurs de Visual Basic 6.0
Nouveautés dans le langage Visual Basic pour les utilisateurs de Visual Basic 6.0
Nouveautés dans les Windows Forms pour les utilisateurs de Visual Basic 6.0
Nouveautés de Visual Basic

Événements et gestion des événements pour les utilisateurs Visual Basic 6.0

Si vous connaissez bien les événements et la gestion des événements dans Visual Basic 6.0, le modèle d'événement dans Visual Basic 2005 peut paraître déconcertant au départ, mais il est finalement plus simple et beaucoup plus puissant.

Différences conceptuelles

Dans Visual Basic 6.0, les événements sont liés à des objets spécifiques et possèdent leur propre code de gestion des événements. Par exemple, sur un formulaire comportant un bouton et un menu, chacun de ces éléments possède son propre événement **Click** ; vous devez écrire du code dans le gestionnaire d'événements pour chacun, même s'ils effectuent exactement la même fonction.

```
' Visual Basic 6.0
Private Sub HelpButton_Click()
    HelpButton.Caption = "Help me!"
End Sub
Private Sub HelpMenu_Click()
    HelpMenu.Caption = "Help me!"
End Sub
```

Dans Visual Basic 2005, les événements sont liés à des gestionnaires d'événements au moyen de délégués, vous permettant de créer un seul gestionnaire d'événements pour plusieurs objets.

VB

```
Private Sub HelpButton_Click(ByVal sender As Object, ByVal e As _
System.EventArgs) Handles HelpButton.Click, HelpMenu.Click
    sender.Text = "Help me!"
End Sub
```

Dans l'exemple ci-dessus, notez que la déclaration de l'événement comporte une clause `Handles` qui définit quels événements seront gérés ; dans ce cas, ce sont les événements **Click** des objets **HelpButton** et **HelpMenu**. Les objets et événements ne doivent pas être obligatoirement du même type? Vous pouvez par exemple avoir un seul gestionnaire d'événements qui gère l'événement **Click** d'un bouton, l'événement **DoubleClick** d'une zone de texte et l'événement **Tick** d'un timer.

Notez également que la déclaration de l'événement contient deux paramètres : `ByVal sender As Object` et `ByVal e As System.EventArgs`. Le premier paramètre, `sender`, fournit une référence à l'objet qui a déclenché l'événement. Le deuxième paramètre, `e`, passe un objet spécifique à l'événement géré. En référençant les propriétés de l'objet (et parfois ses méthodes), vous pouvez obtenir des informations, telles que l'emplacement de la souris pour les événements de souris ou les données transférées pour les événements de glissement-déplacement.

Dans l'exemple suivant, le gestionnaire d'événements **MouseDown** utilise le paramètre `sender` pour déterminer le type d'objet qui a généré l'événement et si l'objet est un **PictureBox**, le paramètre `e` est utilisé pour déplacer une étiquette à l'emplacement où le clic a eu lieu. Pour dupliquer cet exemple, ajoutez deux contrôles **PictureBox** et un contrôle **Label** à un formulaire.

VB

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As _
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown, _
PictureBox1.MouseDown, PictureBox2.MouseDown

    If TypeOf sender Is PictureBox Then
        Label1.Location = sender.Location + e.Location
    Else
        MsgBox("Please click a picture")
    End If
End Sub
```

En général, chaque événement produit un gestionnaire d'événements avec un type event-object pour le second paramètre. Quelques gestionnaires d'événements, tels que ceux des événements **MouseDown** et **MouseUp**, ont le même type d'objet (**MouseEventArgs**) pour leur deuxième paramètre. Dans ce cas, vous pouvez utiliser le même gestionnaire d'événements pour les deux événements.

Pour les événements qui passent différents types d'objets événement, vous devez créer des gestionnaires d'événements séparés. Par exemple, l'événement **TextChanged** d'un contrôle **TextBox** passe l'objet événement **EventArgs** générique et l'événement **MouseDown** passe l'objet événement **MouseEventArgs** plus spécialisé. L'objet **MouseEventArgs** contient des propriétés qui sont spécifiques aux événements de souris tels que **Button**, pour déterminer quel bouton de la souris a été actionné. Ces propriétés ne s'appliquent pas à un contrôle **TextBox** et provoqueraient une erreur si vous essayiez de les référencer.

En plus des différences conceptuelles relatives à la gestion des événements, les noms et comportement de certains événements de plusieurs objets sont différents dans Visual Basic 2005. Pour plus d'informations, consultez Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0.

Voir aussi

Concepts

Vue d'ensemble des gestionnaires d'événements (Windows Forms)

Autres ressources

Création de gestionnaires d'événements dans les Windows Forms

Environnement de développement intégré pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 est totalement intégré à l'environnement de développement intégré (IDE, Integrated Development Environment) de Microsoft Visual Studio 2005, qui diffère de l'IDE Visual Basic 6.0 sur certains points.

Modification de la disposition et des fenêtres

Dans Visual Basic 2005, la disposition standard des fenêtres est différente de celle de Visual Basic 6.0. Si vous préférez la disposition Visual Basic 6.0, vous pouvez la définir en réorganisant manuellement les fenêtres. Votre organisation sera conservée la prochaine fois que vous ouvrirez l'IDE.

Dans Visual Basic 6.0, la valeur par défaut de l'IDE est la disposition MDI (interface multidocument). Vous disposez également de la disposition SDI (interface monodocument). Dans Visual Basic 2005, la disposition par défaut de l'IDE est la disposition **Documents avec onglet** et la disposition SDI n'est plus prise en charge. Pour passer à la disposition MDI, dans le menu **Outils** choisissez **Options** puis, dans **Environnement**, onglet **Général**, sélectionnez **Multidocument (MDI)**.

Dans Visual Basic 6.0, vous pouvez contrôler le comportement de l'ancrage de certaines fenêtres d'outils en choisissant **Options** dans le menu **Outils**. Dans Visual Basic 2005, toutes les fenêtres sont ancrables par défaut. Vous pouvez contrôler leur comportement avec la commande **Ancrable** du menu **Fenêtre**.

Visual Basic 6.0 et Visual Basic 2005 présentent également des différences en termes de fenêtres disponibles.

Remarque
Dans Visual Basic 2005, certaines des fenêtres de débogage sont disponibles uniquement en mode exécution. Lorsque vous basculez en mode Design, les fenêtres sont masquées.

Le tableau suivant répertorie quelques-unes des fenêtres Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	**Visual Basic 2005**
Palette de couleurs	Nouvelle implémentation.
Vue de données	Nouvelle implémentation, remplacée par Explorateur de serveurs
Présentation des formulaires	Nouvelle implémentation.
Explorateur de projets	Explorateur de solutions
Espions	Espion 1, Espion 2, Espion 3, Espion 4

Modifications dans les menus

Dans Microsoft Visual Studio 2005, les commandes de menu sont standardisées pour toutes les langues ; dans certains cas, les commandes de menu Visual Basic 6.0 sont remplacées par des commandes différentes dans Microsoft Visual Studio 2005.

Dans Visual Basic 6.0, toutes les commandes de menu sont visibles à tout moment, mais sont estompées lorsqu'elles se trouvent hors contexte. Dans Visual Basic 2005, le jeu de commandes disponibles varie en fonction du contexte.

Le tableau suivant répertorie les commandes communes de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	**Visual Basic 2005**
Compléments, Gestionnaire de compléments	Outils, Gestionnaire de compléments
Déboguer, Ajouter un espion	Ajouter un espion (menu contextuel uniquement)
Déboguer, Modifier un espion	Nouvelle implémentation. Les espions sont modifiés dans la fenêtre Espion

Déboguer, Exécuter jusqu'au curseur	Exécuter jusqu'au curseur (menu contextuel uniquement)
Déboguer, Définir l'instruction suivante	Définir l'instruction suivante (menu contextuel uniquement)
Déboguer, Afficher l'instruction suivante	Afficher l'instruction suivante (menu contextuel uniquement)
Édition, Compléter le mot	Édition, Intellisense, Compléter le mot
Édition, Rechercher	Édition, Recherche rapide
Édition, Suivant	Édition, Recherche rapide
Édition, Retrait	Édition, Options avancées, Augmenter le retrait de ligne
Édition, Insérer un fichier	Édition, Insérer le fichier comme texte
Édition, Répertorier les constantes	Édition, IntelliSense, Liste des membres
Édition, Répertorier les propriétés/méthodes	Édition, IntelliSense, Liste des membres
Édition, Retrait négatif	Édition, Options avancées, Réduire le retrait de ligne
Édition, Information sur les paramètres	Édition, IntelliSense, Information sur les paramètres
Édition, Infos express	Édition, IntelliSense, Infos express
Édition, Remplacer	Édition, Remplacement rapide
Fichier, Créer NomProjet	Générer, Générer NomProjet
Fichier, Créer le groupe de projets	Générer, Générer une solution
Fichier, Configuration de l'impression	Fichier, Mise en page
Fichier, Fichiers récents	Fichier, Projets récents
Fichier, Supprimer le projet	Édition, Enlever
Fichier, Enregistrer le groupe de projets	Fichier, Enregistrer NomSolution ou Fichier, Enregistrer tout
Fichier, Enregistrer le groupe de projets sous	Fichier, Enregistrer NomSolution sous
Fichier, Enregistrer la sélection	Fichier, Enregistrer les éléments sélectionnés

Format, Ajuster à la grille	Nouvelle implémentation. Contrôlé par l'option d'environnement Aligner sur la grille.
? (Aide), Microsoft sur le Web	Communauté, Centre de développement
Projet, Ajouter un module de classe	Projet, Ajouter une classe
Projet, Ajouter un environnement de données	Nouvelle implémentation. L'environnement de données n'est pas pris en charge.
Projet, Ajouter un rapport de données	Nouvelle implémentation. Les rapports de données ne sont pas pris en charge.
Projet, Ajouter une page DHTML	Nouvelle implémentation. Les pages DHTML ne sont pas prises en charge.
Projet, Ajouter un fichier	Projet, Ajouter un nouvel élément ou Projet, Ajouter un élément existant Remarque Dans Visual Basic 6.0, le fichier proprement dit est ajouté au projet. Dans Visual Basic 2005, une copie du fichier est ajoutée.
Projet, Ajouter une feuille	Projet, Ajouter un formulaire Windows ou Projet, Ajouter un formulaire hérité
Projet, Ajouter une feuille MDI	Projet, Ajouter un formulaire Windows ou Projet, Ajouter un formulaire hérité
Projet, Ajouter une page de propriétés	Nouvelle implémentation. Les pages de propriétés ne sont pas prises en charge.
Projet, Ajouter un document utilisateur	Nouvelle implémentation. Les documents ActiveX ne sont pas pris en charge.
Projet, Ajouter une classe Web	Nouvelle implémentation. Les classes Web ne sont pas prises en charge.
Projet, Composants	Nouvelle implémentation. Les composants sont gérés dans la boîte à outils.
Projet, Références	Nouvelle implémentation. Les références sont gérées dans l'Explorateur de solutions.
Projet, Supprimer élément	Édition, Supprimer **Remarque** La commande "Supprimez l'élément" supprimait l'élément du projet, mais non du disque ; La commande "élimination" supprime l'élément.
Exécuter, Arrêter	Déboguer, Interrompre tout
Exécuter, Fin	Déboguer, Arrêter le débogage
Exécuter, Redémarrer	Déboguer, Redémarrer
Exécuter, Démarrer	Déboguer, Démarrer le débogage
Exécuter, Exécuter avec compilation complète	Nouvelle implémentation. Vous pouvez encore exécuter sans déboguer en appuyant sur CTRL+F5.

Outils, Ajouter une procédure	Nouvelle implémentation. Les procédures sont ajoutées dans l'éditeur de code.
Outils, Éditeur de menus	Nouvelle implémentation. L'Éditeur de menus est remplacé par les composants MainMenu et ContextMenu.
Outils, Attributs de procédure	Nouvelle implémentation.
Outils, SourceSafe	Fichier, Contrôle de code source
Affichage, Pile des appels	Déboguer, Fenêtres, Pile des appels
Affichage, Palette de couleurs	Nouvelle implémentation. Il n'y a pas de fenêtre Palette de couleurs.
Affichage, fenêtre Vue de données	Nouvelle implémentation. Il n'y a pas de fenêtre Vue de données.
Affichage, Définition	Atteindre la définition (menu contextuel uniquement)
Affichage, Fenêtre Présentation des feuilles	Nouvelle implémentation. Il n'y a pas de fenêtre Présentation des feuilles.
Affichage, Fenêtre Exécution	Déboguer, Fenêtres, Immédiat
Affichage, Dernier emplacement	Nouvelle implémentation. Utilisez les signets
Affichage, Fenêtre Variables locales	Déboguer, Fenêtres, Variables locales
Affichage, Objet	Affichage, Concepteur
Affichage, Explorateur de projets	Affichage, Explorateur de solutions
Affichage, Fenêtre Espion	Déboguer, Fenêtres, Espion
Fenêtre, Réorganiser les icônes	Nouvelle implémentation.
Fenêtre, Cascade	Disponible uniquement en mode MDI.
Fenêtre, Mosaïque horizontale	Disponible uniquement en mode MDI.
Fenêtre, Mosaïque verticale	Disponible uniquement en mode MDI.
Fenêtre, Liste des fenêtres	Fenêtres

Modifications de la configuration du clavier

Les configurations du clavier dans Microsoft Visual Studio 2005 sont basées sur un profil de paramètres que vous choisissez la première fois que vous chargez l'IDE. Si vous avez choisi le profil Visual Basic, les configurations du clavier seront les mêmes que dans Visual Basic 6.0. Vous pouvez modifier les configurations du clavier ou les restaurer à leur paramétrage d'origine dans la boîte de dialogue **Outils, Options**. Pour plus d'informations, consultez Paramètres Visual Studio.

Voir aussi

Autres ressources

Personnalisation de l'environnement de développement
Présentation du débogueur
Aide pour les utilisateurs de Visual Basic 6.0

Gestion de projets pour les utilisateurs de Visual Basic 6.0

Les projets Visual Basic 2005 sont similaires à ceux de Visual Basic 6.0. Toutefois, vous devez tenir compte de certaines différences.

Dans cette section

Comportement du projet pour les utilisateurs de Visual Basic 6.0

Décrit les différences générales dans le comportement des projets entre Visual Basic 6.0 et Visual Basic 2005.

Propriétés du projet pour les utilisateurs de Visual Basic 6.0

Décrit les différences de propriétés de projet entre Visual Basic 6.0 et Visual Basic 2005.

Références du projet pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière de gestion des références entre Visual Basic 6.0 et Visual Basic 2005.

Types de projets pour les utilisateurs de Visual Basic 6.0

Décrit les différences de types de projet entre Visual Basic 6.0 et Visual Basic 2005.

Rubriques connexes

Propriétés de projet (Visual Studio)

Décrit le système de projets permettant de gérer vos applications à mesure que vous les écrivez.

Comportement du projet pour les utilisateurs de Visual Basic 6.0

Les projets de Visual Basic 2005 sont semblables aux projets de Visual Basic 6.0. Il existe toutefois quelques différences de comportement. Cette rubrique présente ces différences.

Modèle de projet

Dans Visual Basic 6.0, les projets utilisent un modèle basé sur les références. Le fichier projet contient des références aux éléments de projet en spécifiant leur chemin. Par exemple, lorsque vous ajoutez un fichier texte à un projet, le fichier projet pointe sur l'emplacement du fichier. Lorsque le projet est généré, le fichier texte doit être chargé à partir de cet emplacement.

Visual Basic 2005 utilise un modèle basé sur les dossiers. Tous les éléments de projet sont placés dans la hiérarchie des dossiers du projet. Lorsque vous ajoutez un fichier texte, une copie du fichier est placée dans le dossier du projet. Lorsque le projet est généré, le fichier texte est chargé à partir de cette copie du fichier.

Commande Enlever

Dans Visual Basic 6.0, la commande **Enlever** supprime un élément d'un projet. Toutefois, le fichier reste sur le disque.

Dans Visual Basic 2005, la commande **Enlever** est remplacée par une commande **Supprimer** qui supprime le fichier, à la fois dans le projet et sur le disque. La commande **Exclure du projet** peut être utilisée pour exclure un fichier d'un projet sans l'effacer.

Groupes de projets

Dans Visual Basic 6.0, il est possible d'ajouter plusieurs projets dans l'**Explorateur de projets**, sous forme de groupes de projets.

Dans Visual Basic 2005, l'**Explorateur de solutions** remplace l'**Explorateur de projets**. Les groupes de projets sont remplacés par des solutions. Alors que les groupes de projets ne contenaient que des projets Visual Basic, les solutions peuvent contenir des projets créés avec n'importe quelle combinaison de langages Visual Basic 2005.

Fichiers projet

Dans Visual Basic 6.0, les fichiers projet (.vbp) et les fichiers de groupes de projets (.vbg) sont des fichiers texte que vous pouvez éditer directement dans un éditeur de texte.

Dans Visual Basic 2005, les fichiers projet et solution sont au format XML. Ils ne doivent pas être édités directement.

Voir aussi

Concepts

Propriétés du projet pour les utilisateurs de Visual Basic 6.0

Types de projets pour les utilisateurs de Visual Basic 6.0

Autres ressources

Gestion de projets pour les utilisateurs de Visual Basic 6.0

Propriétés du projet pour les utilisateurs de Visual Basic 6.0

Les propriétés de projet dans Visual Basic 6.0 et Visual Basic 2005 spécifient plusieurs paramètres globaux pour un projet. Nombre des propriétés de projet diffèrent entre Visual Basic 6.0 et Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, les propriétés de projet sont définies dans la boîte de dialogue **Propriétés du projet**.

Dans Visual Basic 2005, les propriétés de projet sont définies dans le **Concepteur de projets**. Pour plus d'informations, consultez Introduction au Concepteur de projets.

Propriétés des composants ActiveX

Dans Visual Basic 6.0, il existe plusieurs propriétés relatives aux composants ActiveX, y compris **Mode d'exécution autonome**, **Modèle de thread**, **Demander le n° de licence**, **Conserver en mémoire**, **Mode de démarrage**, **Serveur distant** et **Compatibilité des versions**.

Dans Visual Basic 2005, le modèle de création de composants est très différent, et ces propriétés de projet n'ont pas d'équivalents directs. Pour plus d'informations, consultez Création de composants pour les utilisateurs de Visual Basic 6.0.

Propriété Mettre à niveau les contrôles ActiveX

Dans Visual Basic 6.0, la propriété de projet **Mettre à niveau les contrôles ActiveX** met automatiquement à jour tous les contrôles ActiveX si vous en avez installé une version plus récente depuis la dernière ouverture du projet.

Dans Visual Basic 2005, bien que vous puissiez utiliser des contrôles ActiveX, il n'existe aucun équivalent pour cette propriété. Le projet utilisera la version du contrôle ActiveX que vous avez référencé lors de son ajout dans le projet.

Propriété Supprimer les informations relatives aux contrôles ActiveX inutilisés

Dans Visual Basic 6.0, la propriété de projet **Supprimer les informations relatives aux contrôles ActiveX inutilisés** élimine toutes les références aux contrôles inutilisés avant la compilation.

Dans Visual Basic 2005, il n'existe aucun équivalent. Les contrôles inutilisés ne sont jamais compilés.

Optimisations du compilateur

Bien que Visual Basic 6.0 et Visual Basic 2005 prennent tous deux en charge des optimisations de compilateur, les compilateurs, et par conséquent les optimisations disponibles, ne sont pas identiques. Pour plus d'informations, consultez Compilateur Visual Basic.

Les optimisations du compilateur Visual Basic 2005 sont définies dans la boîte de dialogue **Paramètres avancés du compilateur**, accessible dans l'onglet **Compiler** du **Concepteur de projets**.

Équivalences des propriétés de projet

La liste suivante présente une comparaison entre les propriétés de projet Visual Basic 6.0 et Visual Basic 2005. Les informations relatives à l'emplacement des propriétés de projet Visual Basic 2005 sont indiquées, en retrait, sous chaque nom de propriété. Notez que toutes les propriétés ne sont pas disponibles pour tous les types de projet.

Visual Basic 6.0	Visual Basic 2005
Incrément automatique	**Incrémenter automatiquement la version à chaque publication** Onglet **Publier** du **Concepteur de projets**.
Arguments de la ligne de commande	**Arguments de la ligne de commande** Onglet **Déboguer** du **Concepteur de projets**.
Options de compilation	Nouvelle implémentation. Pour plus d'informations, consultez Compilateur Visual Basic.

Arguments de la compilation conditionnelle	**Constantes de compilation** La boîte de dialogue **Paramètres avancés du compilateur**, accessible dans l'onglet **Compiler** du **Concepteur de projets**.
Adresse de base DLL	**Adresse de base DLL** La boîte de dialogue **Paramètres avancés du compilateur**, accessible dans l'onglet **Compiler** du **Concepteur de projets**.
Nom du fichier d'aide	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Icon	**Icon** Onglet **Déboguer** du **Concepteur de projets**.
Description du projet	**Description**—boîte de dialogue **Informations de l'assembly**, accessible de l'onglet **Application** du **Concepteur de projets**.
ID de contexte d'aide du projet	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Nom du projet	**Nom de l'assembly** Onglet **Déboguer** du **Concepteur de projets**.
Type de projet	**Type d'application** Onglet **Déboguer** du **Concepteur de projets**.
Serveur distant	Pas d'équivalent.
Supprimer les informations relatives aux contrôles ActiveX inutilisés	Pas d'équivalent.
Demander le n° de licence	Pas d'équivalent.
Conservé en mémoire	Pas d'équivalent.
Mode de démarrage (composants)	Pas d'équivalent.
Objet de démarrage	**Formulaire de démarrage, Objet de démarrage** Onglet **Application** du **Concepteur de projets**.
Modèle de thread	Pas d'équivalent.
Titre	**Titre** La boîte de dialogue **Informations de l'assembly**, accessible de l'onglet **Application** du **Concepteur de projets**.
Mode d'exécution autonome	Pas d'équivalent.
Mettre à niveau les contrôles ActiveX	Pas d'équivalent.
Compatibilité des versions	Pas d'équivalent.

Informations de version	**Société, produit, copyright, marque** La boîte de dialogue **Informations de l'assembly**, accessible de l'onglet **Application** du **Concepteur de projets**.
Numéro de version	**Version de l'assembly** La boîte de dialogue **Informations de l'assembly**, accessible de l'onglet **Application** du **Concepteur de projets**.
Lorsque ce projet démarre	**Action de démarrage** Onglet **Déboguer** du **Concepteur de projets**.

Voir aussi

Concepts

Création de composants pour les utilisateurs de Visual Basic 6.0

Autres ressources

Propriétés de projet (Visual Studio)

Gestion de projets pour les utilisateurs de Visual Basic 6.0

Références du projet pour les utilisateurs de Visual Basic 6.0

Les méthodes d'ajout de références de projet diffèrent entre Visual Basic 2005 et Visual Basic 6.0.

Différences conceptuelles

Ajout de références

Dans Visual Basic 6.0, les références de projets aux bibliothèques de types sont gérées dans la boîte de dialogue **Références**. Une fois qu'une référence est ajoutée, vous pouvez accéder à l'objet dans le code.

Dans Visual Basic 2005, les références sont faites aux assemblys, puis sont ajoutées et supprimées dans le **Concepteur de projets**. Les références peuvent également être créées vers des composants COM. Les références projet-à-projet et les références Web aux services Web ASP.NET sont également prises en charge. Pour plus d'informations, consultez Gestion des références.

Ajout de contrôles

Dans Visual Basic 6.0, les contrôles sont ajoutés à un projet dans la boîte de dialogue **Composants**. Une fois ajoutés, ils apparaissent dans la **boîte à outils**.

Dans Visual Basic 2005, les contrôles sont ajoutés dans la **boîte à outils** et dans la boîte de dialogue **Personnaliser la boîte à outils**. Pour plus d'informations, consultez Comment : ajouter des éléments à la boîte à outils.

Voir aussi

Autres ressources

Références aux espaces de noms et aux composants
Gestion de projets pour les utilisateurs de Visual Basic 6.0

Types de projets pour les utilisateurs de Visual Basic 6.0

Les types de projets que vous pouvez créer avec Visual Basic 2005 sont différents de ceux de Visual Basic 6.0. Dans certains cas, hormis les noms, qui ont changé, les types de projet sont essentiellement identiques. Dans d'autres cas, certains types de projet Visual Basic 6.0 n'ont aucun équivalent direct.

La liste suivante présente une comparaison entre les types de projet Visual Basic 6.0 et leurs équivalents Visual Basic 2005.

Visual Basic 6.0	Visual Basic 2005
Standard EXE	**Application Windows**
ActiveX DLL	**Bibliothèque de classes**
ActiveX EXE	**Bibliothèque de classes**
Contrôle ActiveX	**Bibliothèque de contrôles Windows** **Remarque** Les projets **Bibliothèque de contrôles Windows** créent des contrôles Windows Forms ; la création de contrôles ActiveX n'est plus prise en charge.
Document ActiveX	Aucun équivalent. Visual Basic 2005 peut interagir avec les documents ActiveX.
Application DHTML	Pas d'équivalent. Utilisez une application de site Web ASP.NET Visual Web Developer.
Application IIS (classe Web)	Pas d'équivalent. Utilisez une application de site Web ASP.NET Visual Web Developer.

Voir aussi

Concepts

Modèles de projet par défaut dans Visual Studio

Programmation Web pour les utilisateurs de Visual Basic 6.0

Autres ressources

Gestion de projets pour les utilisateurs de Visual Basic 6.0

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Si les versions antérieures de Visual Basic visaient à créer des applications clientes Microsoft Windows, les premières versions de Visual Basic .NET 2002 ont été conçues pour également créer des applications de service Web XML. À cette fin, Visual Basic génère du code managé pour le Common Language Runtime. Cette nouvelle perspective implique un certain nombre de modifications liées au langage proprement dit.

Les modifications en Visual Basic visent à :

- simplifier le langage et le rendre plus cohérent ;
- ajouter de nouvelles fonctionnalités demandées par les utilisateurs ;
- rendre le code facile à lire et à gérer ;
- aider les programmeurs à ne pas introduire d'erreurs de code ;
- rendre les applications plus robustes et plus faciles à déboguer.

Les rubriques suivantes décrivent les modifications apportées au langage par rapport aux versions antérieures de Visual Basic. Pour obtenir un examen des nouvelles fonctionnalités du langage de Visual Basic, consultez Nouveautés du langage Visual Basic.

Dans cette section

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique des éléments de programmation qui ont changé ou qui ne sont plus pris en charge en Visual Basic et indique les éléments de remplacement.

Tableaux pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent les modifications relatives aux limites de tableau, aux déclarations de la taille et à ReDim, instruction (Visual Basic).

Types de données pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent les modifications relatives à la déclaration, à l'utilisation et à la conversion des types de données, y compris le Integer, type de données (Visual Basic) et le type de données universel.

Modifications des déclarations pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent des modifications relatives à la syntaxe de déclaration, à la longueur de chaîne, aux structures et à la portée des variables au sein de blocs.

Modifications des fonctions pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent les modifications relatives à la fonction Format, aux fonctions liées aux dates et heures, ainsi qu'aux fonctions qui incluent le suffixe de type de données String (Visual Basic) (**$**).

Modifications diverses du langage pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent des modifications relatives aux opérateurs et à la gestion de fichiers.

Modifications des procédures pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent les modifications relatives au passage de paramètres, aux séquences d'appel de procédure et aux déclarations de procédure.

Modifications des propriétés pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent des modifications relatives aux propriétés par défaut, aux procédures de propriété et aux arguments de propriété.

Modifications du flux de contrôle pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent des modifications relatives au flux d'exécution et à la nouvelle prise en charge de la gestion structurée des exceptions.

Programmation orientée objet pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent des modifications relatives à la création d'objets, à la compatibilité binaire, aux interfaces et aux classes.

Rubriques connexes

Nouveautés du langage Visual Basic

Inclut des informations sur la continuation de boucle, la mise à disposition de ressources garanties, les propriétés d'accès mixte, les types de données non signés, la surcharge d'opérateur, les types génériques et la conformité Spécification CLS (Common Language Specification) (CLS).

Aide pour les utilisateurs de Visual Basic 6.0

Traite des modifications relatives à Visual Basic qui incluent notamment la fonctionnalité Web, les projets, les formulaires, les constantes, les méthodes **Circle**, **Line** et **Pset**, ainsi que l'environnement de développement intégré (IDE).

Référence Visual Basic

Constitue une référence complète du langage Visual Basic et de ses éléments runtime.

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Visual Basic 2005 modifie ses méthodes de prise en charge de divers éléments de programmation, principalement en ce qui concerne l'interopérabilité avec le Common Language Runtime. La plupart des éléments Visual Basic 6.0 sont renommés, reclassifiés ou associés à d'autres éléments de programmation pour Visual Basic 2005. Plusieurs éléments ne sont plus pris en charge, car le Common Language Runtime (CLR) inclut désormais des fonctionnalités qui les rendent inutiles. Pour plus d'informations, consultez Common Language Runtime.

Pour obtenir des informations supplémentaires sur les modifications apportées à Visual Basic, y compris l'environnement de développement intégré (IDE), les fonctionnalités Web, les projets, les formulaires, les constantes et les méthodes **Circle**, **Line** et **Pset**, consultez Aide pour les utilisateurs de Visual Basic 6.0.

Éléments modifiés

Le tableau suivant répertorie les éléments de programmation modifiés ainsi que leurs équivalents.

Élément de programmation Visual Basic 6.0	**Équivalent Visual Basic 2005**	**Espace de noms, classe ou emplacement de la bibliothèque runtime**
Fonction **Abs**	Méthode Abs	System, espace de noms, Math, classe
Fonction **AscB**	Fonction Asc	Membres de la bibliothèque Runtime Visual Basic, module **Strings**

Expression de mot clé **As Any**	N'est pas pris en charge dans Visual Basic 2005. Consultez Syntaxe de déclaration pour les utilisateurs de Visual Basic 6.0.	Non applicable
Fonction **Atn**	Méthode Atan	System, espace de noms, Math, classe
Propriété **Calendar**	Propriété CurrentCulture	System.Globalization, espace de noms, CultureInfo, classe
Instruction **ChDir**	ChDir, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **ChDrive**	ChDrive, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Chr$, Fonctions **ChrB**	Fonction Chr	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instruction **Close**	FileClose, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Fonction **Cos**	Méthode Cos	System, espace de noms, Math, classe
Currency (type de données)	Type de données décimal, consultez Modifications des types de données pour les utilisateurs de Visual Basic 6.0	Membres de la bibliothèque runtime Visual Basic, VariantType, énumération
Fonction **CVDate**	DateValue, fonction	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**

Fonction **CVErr**	Error, instruction	Non applicable
Fonction **Date**, instruction **Date**	Now, propriété, Today, propriété	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**
Fonction **Date$**	DateString, propriété	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**
Méthode **Debug.Assert**	Méthodes Assert, Fail	System.Diagnostics, espace de noms, Debug, classe

Méthode **Debug.Print**	Méthodes Write, WriteIf, WriteLine et WriteLineIf	System.Diagnostics, espace de noms, Debug, classe
Instructions **Deftype**	N'est pas pris en charge dans Visual Basic 2005. Consultez Modifications des types de données pour les utilisateurs de Visual Basic 6.0	Non applicable
Instruction **DeleteSetting**	DeleteSetting, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Interaction**
Fonction **DoEvents**	Méthode DoEvents	System.Windows.Forms, espace de noms, Application, classe

Mot clé **Empty**	Rien	Non applicable
Opérateur **Eqv**	Opérateur = ; consultez Opérateur booléen pour les utilisateurs de Visual Basic 6.0	Non applicable
Fonction **Exp**	Méthode Exp	System, espace de noms, Math, classe
Instruction **FileCopy**	FileCopy, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **Get**	FileGet, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

GoSub (instruction)	N'est pas pris en charge dans Visual Basic 2005 ; utilisez l'instruction return. Consultez Instruction de contrôle pour les utilisateurs de Visual Basic 6.0.	Non applicable
Opérateur **Imp**	N'est pas pris en charge dans Visual Basic 2005. Consultez les opérateurs Différent de et Ou ainsi que Opérateur booléen pour les utilisateurs de Visual Basic 6.0.	Non applicable
Événement **Initialize**	N'est pas pris en charge dans Visual Basic 2005 ; utilisez **Sub New**. Consultez Modifications de Class_Initialize pour les utilisateurs de Visual Basic 6.0 et Utilisation de constructeurs et de destructeurs.	Non applicable

Instructions **Input #**, **Input $**, fonctions **Input $**, **InputB**, **InputB $**	Input, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Propriété **Instancing**	N'est pas pris en charge dans Visual Basic 2005. Pour les déclarations au niveau de la classe, consultez Private (Visual Basic) et Public (Visual Basic). Pour les déclarations au niveau de la procédure, consultez Shared (Visual Basic).	Non applicable
Fonction **InStrB**	InStr, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**

Fonction **IsEmpty**	IsNothing, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Information**
Fonction **IsMissing**	N'est pas pris en charge dans Visual Basic 2005. Consultez Déclaration de procédure pour les utilisateurs de Visual Basic 6.0.	Non applicable
Fonction **IsNull**	Fonction IsDBNull	Membres de la bibliothèque Runtime Visual Basic, module **Information**
Fonction **IsObject**	IsReference, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Information**
Instruction **Kill**	Kill, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Fonction **LCase$**	LCase, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Fonctions **Left$**, **LeftB**, **LeftB$**	Left, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Fonction **LenB**	Len, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instructions d'assignation **Let**, **Set**	N'est pas pris en charge dans Visual Basic 2005 ; la nouvelle instruction **Set** n'a aucun rapport avec l'ancienne. Consultez Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0.	Non applicable

Instruction **LineInput #**	LineInput, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **Load**	Mot clé New (Visual Basic), méthode Show, événement Load	Membres de la bibliothèque runtime Visual Basic, espace de noms System.Windows.Forms
Instruction **Lock**	Lock, Unlock, fonctions	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Fonction **Log**	Méthode Log	System, espace de noms, Math, classe
Instructions **LSet**, **RSet**	Fonction LSet, PadRight, PadLeft ; consultez Modifications des types de données pour les utilisateurs de Visual Basic 6.0	Membres de la bibliothèque Runtime Visual Basic, module **Strings** et espace de noms System, classe String

Fonction **LTrim$**	Fonction LTrim	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Fonction **MidB**	Mid, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instruction **MidB**	Mid, instruction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instruction **MkDir**	MkDir, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **Name**	Rename, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Fonction **Now**	Now, propriété	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**
Mot clé **Null**	Rien	Non applicable
Fonction **Oct$**	Oct, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Conversion**
Construction **On...GoSub**	N'est pas pris en charge dans Visual Basic 2005 ; utilisez l'instruction Select...Case. Consultez Instruction de contrôle pour les utilisateurs de Visual Basic 6.0.	Non applicable
Construction **On...GoTo**	N'est pas pris en charge dans Visual Basic 2005 ; utilisez l'instruction Select...Case. Consultez Instruction de contrôle pour les utilisateurs de Visual Basic 6.0.	Non applicable

Instruction **Open**	FileOpen, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **OptionBase**	N'est pas pris en charge dans Visual Basic 2005. Consultez Limites de tableau pour les utilisateurs de Visual Basic 6.0.	Non applicable
Instruction **OptionPrivateModule**	N'est pas pris en charge dans Visual Basic 2005 ; utilisez l'instruction Module.	Non applicable
Instruction **Print #**	Print, PrintLine, fonctions	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Instructions **PropertyGet**, **PropertyLet**, **PropertySet**	N'est pas pris en charge dans Visual Basic 2005. Consultez Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0.	Non applicable
Instruction **Put**	FilePut, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **Reset**	Reset, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Fonctions **Right$**, **RightB**	Right, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instruction **RmDir**	RmDir, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Fonction **Round**	Méthode Round	System, espace de noms, Math, classe
Instructions **RSet**, **LSet**	Fonction RSet ; consultez Modifications des types de données pour les utilisateurs de Visual Basic 6.0	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Fonction **RTrim$**	Fonction RTrim	Membres de la bibliothèque Runtime Visual Basic, module **Strings**

Instruction **SaveSetting**	SaveSetting, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Interaction**
Méthode **Scale**	N'est pas pris en charge dans Visual Basic 2005.	Non applicable
Instructions d'assignation **Set**, **Let**	N'est pas pris en charge dans Visual Basic 2005 ; la nouvelle instruction **Set** n'a aucun rapport avec l'ancienne. Consultez Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0	Non applicable
Instruction **SetAttr**	SetAttr, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Fonction **Sgn**	Méthode Sign	System, espace de noms, Math, classe
Fonction **Sin**	Méthode Sin	System, espace de noms, Math, classe
Fonction **Sqr**	Méthode Sqrt	System, espace de noms, Math, classe
Fonction **String**	Constructeur String ; consultez Fonction String ($) pour les utilisateurs de Visual Basic 6.0	System, espace de noms, String, classe
Fonctions **String ($)**	N'est pas pris en charge dans Visual Basic 2005. Consultez Fonction String ($) pour les utilisateurs de Visual Basic 6.0.	Non applicable

Événement **Terminate**	N'est pas pris en charge dans Visual Basic 2005 ; utilisez **Sub Dispose** et **Sub Finalize**. Consultez Utilisation de constructeurs et de destructeurs.	Non applicable
Fonction **Time**, instruction **Time**	TimeOfDay, propriété ; consultez DateTime, structure, Date, type de données	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**
Fonction **Time$**	TimeString, propriété	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**
Fonction **Timer**	Timer, propriété	Membres de la bibliothèque Runtime Visual Basic, module **DateAndTime**

Fonction **Trim$**	LTrim, RTrim et Trim, fonctions	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instruction **Type**	N'est pas pris en charge dans Visual Basic 2005 ; utilisez l'instruction Structure. Consultez Déclaration de structure pour les utilisateurs de Visual Basic 6.0.	Non applicable
Fonction **UCase$**	UCase, fonction	Membres de la bibliothèque Runtime Visual Basic, module **Strings**
Instruction **Unlock**	Lock, Unlock, fonctions	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Variant (type de données)	Type de données Object, consultez Modifications du type de données universel pour les utilisateurs de Visual Basic 6.0	Non applicable

Mot clé **Wend**	Instructions While...End et Instruction End ; consultez Instruction de contrôle pour les utilisateurs de Visual Basic 6.0	Non applicable
Instruction **Width#**	FileWidth, fonction	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**
Instruction **Write#**	Write, WriteLine, fonctions	Membres de la bibliothèque Runtime Visual Basic, module **FileSystem**

Voir aussi

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0
Aide pour les utilisateurs de Visual Basic 6.0
Common Language Runtime

Tableaux pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux tableaux entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Limites de tableau pour les utilisateurs de Visual Basic 6.0

Traite des modifications apportées aux limites de tableau et aux limites par défaut.

Déclaration de la taille des tableaux pour les utilisateurs de Visual Basic 6.0

Décrit les modifications apportées à la spécification des tailles de tableau.

Instruction ReDim pour les utilisateurs de Visual Basic 6.0

Traite des modifications apportées aux déclarations comprenant l'instruction **ReDim**.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique des éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Limites de tableau pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 introduit certaines mises à jour concernant la déclaration des limites de tableau, en vue d'assurer l'interopérabilité avec les tableaux issus d'autres langages de programmation.

Visual Basic 6.0

Dans Visual Basic 6.0, la limite inférieure par défaut des dimensions d'un tableau est égale à 0. Vous pouvez remplacer cette valeur par 1 avec l'instruction **Option Base**. Vous pouvez également substituer la limite inférieure par défaut dans des déclarations de tableau individuelles.

Si vous gardez la limite par défaut, à savoir 0, le nombre d'éléments figurant dans le tableau est égal à la limite supérieure plus un. La déclaration suivante réserve 21 éléments pour le tableau `Weight`:

VB

```
Dim Weight(20) As Single
```

Visual Basic 2005

Dans Visual Basic 2005, la limite inférieure des dimensions d'un tableau est égale à 0 et vous ne pouvez pas la déclarer différemment. L'instruction **Option Base** n'est pas prise en charge.

Le nombre spécifié pour chaque dimension dans la déclaration est la limite supérieure, et le décompte des éléments initial est égal à la limite supérieure plus un. La déclaration de l'exemple précédent réserve 21 éléments pour le tableau `Weight`, avec des indices de 0 à 20.

Vous pouvez également indiquer un *tableau de longueur zéro*, qui ne contient pas d'élément, en déclarant que l'une de ses limites supérieures a la valeur –1.

Voir aussi

Référence

Dim, instruction (Visual Basic)

UBound, fonction (Visual Basic)

Concepts

Syntaxe de déclaration pour les utilisateurs de Visual Basic 6.0

Déclaration de la taille des tableaux pour les utilisateurs de Visual Basic 6.0

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Autres ressources

Tableaux dans Visual Basic

Déclaration de la taille des tableaux pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 met à jour la déclaration de la taille des tableaux en vue d'assurer l'interopérabilité avec le Common Language Runtime.

Visual Basic 6.0

Visual Basic 6.0 vous permet de spécifier la taille d'un tableau dans sa déclaration, comme l'illustre d'exemple suivant :

```
Dim Month(0 To 11) As Integer
```

Le tableau possède alors une taille fixe, qui ne peut pas être modifiée à l'aide de l'instruction **ReDim**.

Visual Basic 2005

Dans Visual Basic 2005, un tableau n'a pas de taille permanente. L'exemple précédent peut être réécrit, de sorte à utiliser l'une ou l'autre des déclarations suivantes :

VB

```
Dim Month(11) As Integer   ' Reserves 12 elements -- (0) through (11).
```

VB

```
Dim Month() As Integer = New Integer(11) {}
```

Ces deux déclarations sont équivalentes. Elles spécifient toutes deux une taille initiale, que vous pouvez modifier au cours de l'exécution à l'aide de l'instruction **ReDim**. Pour initialiser les éléments, vous pouvez utiliser la syntaxe suivante :

VB

```
Dim Month() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Si vous indiquez –1 pour l'une des dimensions, le tableau ne contient pas d'élément. Une instruction **ReDim** peut faire passer un tableau de l'état vide à non vide, et inversement.

Nombre de dimensions

Bien que la taille d'un tableau puisse être modifiée dans Visual Basic 2005, le nombre de dimensions doit rester fixe. L'exemple suivant déclare un tableau à trois dimensions :

VB

```
Dim Point(,,) As Double
```

L'instruction **ReDim** permet de définir ou de modifier la taille de chaque dimension, mais le tableau garde toujours ses trois dimensions, ni plus ni moins.

Voir aussi

Référence

Dim, instruction (Visual Basic)
New (Visual Basic)
ReDim, instruction (Visual Basic)

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Autres ressources

Tableaux dans Visual Basic

Instruction ReDim pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 effectue des mises à jour concernant les déclarations avec l'instruction **ReDim** en vue d'une simplification et d'une amélioration de la lisibilité.

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez utiliser l'instruction **ReDim** comme instruction de déclaration initiale d'un tableau dynamique. Il n'est pas nécessaire de déclarer le tableau à un quelconque autre endroit.

Visual Basic 6.0 vous permet également de modifier le rang d'un tableau avec **ReDim**, comme l'illustre l'exemple suivant :

```
Dim X(10) As Single
ReDim X(10, 10)
```

Visual Basic 2005

Dans Visual Basic 2005, vous ne pouvez pas utiliser **ReDim** comme une déclaration. Vous devez déclarer un tableau de la même manière que toute autre variable, à l'aide de l'instruction **Dim** ou d'une instruction équivalente, avant de pouvoir l'intégrer dans une instruction **ReDim**.

Vous ne pouvez pas modifier le rang d'un tableau dans Visual Basic 2005. Le code de l'exemple précédent génère une erreur.

Voir aussi

Référence

ReDim, instruction (Visual Basic)

Dim, instruction (Visual Basic)

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Types de données pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux types de données entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Décrit les changements liés à la déclaration, à l'utilisation et à la conversion des types de données.

Type de données Integer pour les utilisateurs de Visual Basic 6.0

Traite des changements liés au type de données Integer.

Modifications du type de données universel pour les utilisateurs de Visual Basic 6.0

Traite des changements intervenus sur le plan du type de données universel.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique des éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, des mises à jour concernant les types de données ont été apportées en vue d'assurer l'interopérabilité avec d'autres langages de programmation et avec le Common Language Runtime. Les modifications sont liées à la déclaration, à l'utilisation et à la conversion des types de données.

Visual Basic 6.0

Dans Visual Basic 6.0, utilisez les instructions **Deftype** (**DefBool**, **DefByte**, **DefCur**, **DefDate**, **DefDbl**, **DefDec**, **DefInt**, **DefLng**, **DefObj**, **DefSng**, **DefStr** et **DefVar**) pour définir le type par défaut d'une variable.

Le type de données **Currency** vous permet d'effectuer des calculs impliquant de l'argent ainsi que des calculs en virgule fixe.

Une **Date** est stockée sous un format **Double** utilisant huit octets.

Si deux variables **Variant** contenant des entiers sont multipliées, un dépassement de capacité entraîne la conversion du type de données du résultat en **Double**.

Utilisez les instructions **LSet** et **RSet** pour copier une variable d'un certain type défini par l'utilisateur vers une autre variable d'un autre type défini par l'utilisateur.

Visual Basic 2005

Les instructions **Deftype** ne sont pas prises en charge dans Visual Basic 2005. Le type de données **Currency** n'est pas pris en charge non plus. Pour tous les calculs et variables monétaires, utilisez plutôt le nouveau type de données **Decimal**, capable de gérer un nombre supérieur de chiffres de part et d'autre de la virgule décimale. **Decimal** est également directement pris en charge par le Common Language Runtime.

Dans Visual Basic 2005, le type de données **Date** utilise le type de données DateTime du Common Language Runtime, qui est une valeur entière codée sur huit octets. En raison de ces représentations différentes, il ne peut y avoir de conversion implicite entre les types de données **Date** et **Double**. Pour passer de **Double** à la représentation Visual Basic 6.0 de **Date**, utilisez les méthodes ToOADate et FromOADate de la structure **DateTime** dans l'espace de noms System.

Si la multiplication de deux variables **Object** contenant des entiers aboutit à un dépassement de capacité, le résultat est converti dans le type de données à 64 bits **Long**.

Vous ne pouvez pas employer **LSet** et **RSet** pour assigner un type de données à un autre. Cela implique une opération de type risquée (particulièrement avec les structures) qui pourrait engendrer du code non vérifiable.

Voir aussi

Référence

Decimal, type de données (Visual Basic)
Type de données date (Visual Basic)
Val, fonction
Fonctions de conversion de types de données
Liste des types de données (Visual Basic)
Double, type de données (Visual Basic)
Long, type de données (Visual Basic)
DateTime
TimeSpan

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Type de données Integer pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic, des mises à jour concernant les types de données entiers ont été apportées en vue d'assurer l'interopérabilité avec d'autres langages de programmation et avec le Common Language Runtime.

Le tableau suivant fournit les types entiers Visual Basic 6.0 et les types entiers Visual Basic 2005 correspondants.

Taille de l'entier	Type Visual Basic 6.0 et caractère de type d'identificateur	Type et caractère de type d'identificateur dans Visual Basic 2005	Type Common Language Runtime (CLR)
8 bits, signé	(aucun)	SByte (aucun)	System.SByte
8 bits, non signé	**Byte** (aucun)	Byte (aucun)	System.Byte
16 bits, signé	**Integer** (**%**)	Short (aucun)	System.Int16
16 bits, non signé	(aucun)	UShort (aucun)	System.UInt16
32 bits, signé	**Long** (**&**)	Integer (**%**)	System.Int32
32 bits, non signé	(aucun)	UInteger (aucun)	System.UInt32
64 bits, signé	(aucun)	Long (**&**)	System.Int64
64 bits, non signé	(aucun)	ULong (aucun)	System.UInt64

Dans les systèmes 32 bits, les opérations sur les entiers 32 bits sont plus rapides que les opérations sur les entiers 16 bits ou 64 bits. Cela signifie que **Integer** et **UInteger** sont les types numériques les plus effectifs et les plus importants dans Visual Basic 2005. Vous pouvez améliorer les performances de vos applications en remplaçant les déclarations **Long** en déclarations **Integer** lors de la migration vers Visual Basic 2005.

Remarque

Si vous utilisez des composants créés sur des plates-formes autres que Visual Basic 2005, vous devez veiller à ce que vos types de données correspondent à ceux de ces composants. Par exemple, si vous utilisez une instruction **Declare** pour faire référence à une procédure externe créée dans Visual Basic 6.0 et si la procédure définit un argument **Integer** (deux octets dans Visual Basic 6.0), vous devez identifier cet argument en tant que **Short** dans l'instruction **Declare**, car c'est le type entier de deux octets dans Visual Basic 2005.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Declare, instruction

Concepts

Caractères de type

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications du type de données universel pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, des mises à niveau concernant le type de données universel ont été effectuées en vue d'assurer l'interopérabilité avec le Common Language Runtime.

Visual Basic 6.0

Dans Visual Basic 6.0, **Variant** est le type de données universel. Cela signifie qu'il est possible de stocker des données de n'importe quel type dans une variable de type **Variant**.

Visual Basic 2005

Dans Visual Basic 2005, **Object** est le type de données universel. Une variable de type **Object** peut contenir des données de n'importe quel type. Le type **Variant** n'est pas pris en charge, et toutes ses fonctionnalités sont désormais offertes par **Object**.

Variant est encore un mot réservé dans Visual Basic 2005, bien qu'il ne soit pas utilisé d'un point de vue syntaxique. Cela permet d'éviter toute confusion avec son ancienne signification.

La fonction **VarType** retourne un membre de l'énumération **VariantType** qui fournit la classification de type de données d'une variable objet. Vous pouvez également utiliser des classes de l'espace de noms System pour obtenir des informations sur le type de données numériques d'une instance **Object**, comme l'illustre le code suivant :

VB

```
' Visual Basic 2005
Dim SomeObj As New Object
' ... ... ... ...  SomeObj is assigned some value during processing.
' ... ... ... ...  Now we want to find out the data type of SomeObj.
Dim Dtype As Integer      ' To hold numeric data type result.
Dtype = Type.GetTypeCode(SomeObj.GetType())
```

Voir aussi

Référence

Object, type de données
VarType, fonction (Visual Basic)
VariantType, énumération
System
Object
Type

Concepts

Types de données en Visual Basic
Types valeur et types référence
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications des déclarations pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux déclarations entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Syntaxe de déclaration pour les utilisateurs de Visual Basic 6.0

Traite de la déclaration de plusieurs variables et de la déclaration de types de retour.

Déclaration de la longueur des chaînes pour les utilisateurs de Visual Basic 6.0

Traite de la spécification de la longueur des chaînes.

Déclaration de structure pour les utilisateurs de Visual Basic 6.0

Traite des structures, également connues sous le nom de types définis par l'utilisateur.

Portée de variable pour les utilisateurs de Visual Basic 6.0

Traite de la portée des variables à l'intérieur d'un bloc.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique d'éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Syntaxe de déclaration pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 présente plusieurs modifications liées à la déclaration des éléments de programmation.

Déclaration de variables multiples

Dans Visual Basic 2005, la déclaration simultanée de plusieurs variables a été revue dans un but de simplification.

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez déclarer des variables de différents types dans la même instruction, mais vous devez spécifier le type de données de chaque variable, sans quoi celle-ci sera de type **Variant** par défaut. L'exemple suivant illustre plusieurs déclarations et leurs types de données résultants :

```
Dim I, J As Integer             ' I is Variant, J is Integer.
Dim L As Integer, M As Integer  ' L is Integer, M is Integer.
Dim N As Integer, X As Double   ' N is Integer, X is Double.
```

Visual Basic 2005

Dans Visual Basic 2005, vous pouvez déclarer plusieurs variables du même type de données sans avoir à répéter le mot clé qui précise le type. Voici les déclarations équivalentes à celles de l'exemple précédent :

VB

```
Dim I                           ' I is Object.
Dim J As Integer                ' J is Integer.
```

Ou

VB

```
Dim I As Object, J As Integer   ' I is Object, J is Integer.
Dim L, M As Integer             ' L is Integer, M is Integer.
Dim N As Integer, X As Double   ' N is Integer, X is Double.
```

Déclaration de procédures externes

Visual Basic 6.0

Dans Visual Basic 6.0, lorsque vous déclarez une référence à une procédure externe à l'aide de l'instruction **Declare**, vous pouvez spécifier **As Any** pour le type de données de tous les arguments et pour le type de retour. Les mots clés **As Any** désactivent la vérification de type et permettent de passer ou de retourner tout type de données.

Visual Basic 2005

Visual Basic 2005 ne prend pas en charge le mot clé **Any**. Dans une instruction **Declare**, vous devez déclarer spécifiquement le type de données de tous les arguments et de leur valeur de retour si **Option Strict** est **On**. La sécurité du type s'en trouve ainsi améliorée. Vous pouvez surcharger la déclaration de la procédure en fonction des divers types de données d'argument. Une surcharge ne peut pas être appliquée uniquement aux types de retour, mais vous pouvez activer **Option Strict Off** ou utiliser les surcharges de type d'argument pour changer le type de retour.

Déclaration d'étiquette de ligne

Visual Basic 6.0

Dans Visual Basic 6.0, un numéro de ligne peut précéder directement une instruction de la même ligne, sans utiliser de caractère de délimitation.

Visual Basic 2005

Visual Basic 2005 nécessite que chaque étiquette de ligne soit suivie par un signe deux-points (**:**). Une instruction peut éventuellement suivre les deux-points (:) sur la même ligne ; l'étiquette de ligne et les deux-points (:) peuvent également se trouver seuls sur la ligne.

Voir aussi

Référence

Declare, instruction

Option Strict, instruction

Concepts

Modifications du type de données universel pour les utilisateurs de Visual Basic 6.0

Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0

Types de données en Visual Basic

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Déclaration de la longueur des chaînes pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 met à jour la déclaration de la longueur de chaîne en vue d'assurer l'interopérabilité avec le Common Language Runtime.

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez spécifier la longueur d'une chaîne dans sa déclaration. La chaîne possède dès lors une longueur fixe, comme l'illustre l'exemple suivant :

```
Dim Name As String * 30
```

Visual Basic 2005

En Visual Basic 2005, une chaîne ne peut pas être déclarée comme ayant une longueur fixe, sauf si l'attribut de la classe VBFixedStringAttribute est utilisé dans la déclaration. Le code de l'exemple précédent génère une erreur.

Vous devez déclarer une chaîne sans lui attribuer de longueur. Lorsque le code assigne une valeur à la chaîne, la longueur de cette valeur détermine celle de la chaîne, comme l'illustre l'exemple suivant :

VB

```
Dim Name As String
' ...
Name = "Name is now 30 characters long" ' Length can be changed later.
```

Voir aussi

Référence

String, type de données (Visual Basic)
Dim, instruction (Visual Basic)

Concepts

Attributs utilisés dans Visual Basic
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Déclaration de structure pour les utilisateurs de Visual Basic 6.0

Visual Basic considère que les structures et les types définis par l'utilisateur sont des éléments de programmation du même type. Visual Basic 2005 met à jour la déclaration des structures à des fins d'unification et pour une meilleure lisibilité.

Visual Basic 6.0

Dans Visual Basic 6.0, une structure est déclarée à l'aide de la construction **Type ... End Type**. La structure et ses membres présentent par défaut un accès public. Une déclaration d'accès explicite est facultative. L'exemple suivant montre une déclaration de structure valide :

```
Type Employee

EmpNumber As Integer ' Defaults to Public access.

EmpOfficePhone As String

EmpHomePhone As String ' Cannot be declared Private inside Type.

End Type
```

Visual Basic 2005

Dans Visual Basic 2005, l'instruction **Type** n'est pas prise en charge. Vous devez déclarer les structures à l'aide de l'instruction **Structure** dans une construction **Structure ... End Structure**. Tous les membres d'une structure doivent posséder un modificateur d'accès, qui peut être **Public**, **Friend** ou **Private**. Vous pouvez également employer l'instruction **Dim**, qui dispose par défaut de l'accès public. La structure de l'exemple précédent peut être déclarée de la manière suivante :

VB

```
Structure Employee
  Public EmpNumber As Integer    'Must declare access, even if Public.
  Dim EmpOfficePhone As String   'Still defaults to Public access.
  Private EmpHomePhone As String 'Can be made Private inside Structure.
End Structure
```

Dans Visual Basic 2005, la syntaxe des structures et des classes est unifiée. Les structures prennent en charge la plupart des fonctionnalités des classes, y compris les méthodes.

Voir aussi

Référence

Structure, instruction
Dim, instruction (Visual Basic)
Public (Visual Basic)
Private (Visual Basic)

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Portée de variable pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 met à jour la portée des variables locales de façon à prendre en charge la portée de bloc et à améliorer la programmation structurée.

Visual Basic 6.0

Dans Visual Basic 6.0, toute variable déclarée à l'intérieur d'une procédure possède une portée de procédure et est donc accessible partout ailleurs dans la même procédure. Si la variable est déclarée à l'intérieur d'un bloc, c'est-à-dire un ensemble instructions terminées par une instruction **End**, **Loop** ou **Next**, elle demeure accessible à l'extérieur du bloc.

L'exemple suivant illustre la portée de procédure, à l'endroit où le bloc est une boucle **For** :

```
For I = 1 To 10

Dim N As Long = 0

' N has procedure scope although it was declared within a block.

N = N + Incr(I)

Next I

W = Base ^ N

' N is still visible outside the block it is declared in.
```

Visual Basic 2005

Dans Visual Basic 2005, une variable déclarée à l'intérieur d'un bloc possède une portée de bloc et n'est pas accessible à l'extérieur du bloc. L'exemple précédent peut être réécrit de la manière suivante :

VB

```
Dim N As Long = 0
' N is declared outside the block and has procedure scope.
For I As Integer = 1 To 10
    ' I is declared by the For statement and therefore within the block.
    N = N + Incr(I)
Next I
w = Base ^ N
' N is visible outside the block but I is not.
```

Étant donné que l'instruction **For** déclare `I` comme partie intégrante du bloc **For**, `I` possède uniquement une portée de bloc.

Cependant, la durée de vie d'une variable reste celle de la procédure complète, ce qui est vrai, que la variable possède une portée de bloc ou une portée de procédure. Si vous déclarez une variable à l'intérieur d'un bloc, et si vous entrez dans ce bloc à plusieurs reprises au cours de la durée de vie de la procédure, il est conseillé d'initialiser la variable afin d'éviter la présence de valeurs inattendues.

Voir aussi

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications des fonctions pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux fonctions entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Fonction Format pour les utilisateurs de Visual Basic 6.0

Traite des modifications de formats de date/d'heure, numériques et de chaîne apportées à la fonction **Format**.

Date et Time pour les utilisateurs de Visual Basic 6.0

Traite des modifications apportées aux fonctions **Date**, **Date$**, **Time** et **Time$**.

Fonction String ($) pour les utilisateurs de Visual Basic 6.0

Traite des fonctions qui incluent le suffixe $ (**String**).

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique des éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Fonction Format pour les utilisateurs de Visual Basic 6.0

La fonction **Format** de Visual Basic 2005 est désormais conforme à la spécification Common Language Runtime (CLR) de mise en forme des données. Pour plus d'informations sur la mise en forme des données dans le .NET Framework, consultez Mise en forme des types.

Les sections suivantes décrivent en détail les modifications apportées par Visual Basic 2005 aux formats Date/Heure, numériques et de chaîne définis par l'utilisateur.

Formats Date/Heure définis par l'utilisateur

Visual Basic 6.0

Dans Visual Basic 6.0, pour afficher une date courte ou longue, vous utilisez le spécificateur de format "ddddd" ou "dddddd". Les spécificateurs **DayOfWeek** ("w") et **WeekOfYear** ("ww") affichent le jour considéré comme étant le premier jour de la semaine et la semaine considérée comme étant la première semaine de l'année. Le caractère "m" minuscule affiche le mois sous la forme d'un nombre sans zéro non significatif. Le spécificateur **Quarter** ("q") affiche le trimestre de l'année sous la forme d'un nombre compris entre 1 et 4.

Pour afficher les minutes sous la forme d'un nombre avec ou sans zéros non significatifs, vous utilisez soit le spécificateur de format "Nn", soit le spécificateur de format "N". Les caractères "Hh" affichent l'heure sous la forme d'un nombre avec zéros non significatifs, et "ttttt" affiche l'heure sous la forme d'une heure complète. Pour afficher soit un "A" majuscule, soit un "P" majuscule avec toute heure avant ou après midi, vous utilisez "AM/PM", "am/pm", "A/P", "a/p" ou "AMPM".

Le spécificateur de date/heure courte ("c") affiche une date et une heure au format "ddddd ttttt".

Visual Basic 2005

Dans Visual Basic 2005, "ddddd" et "dddddd" se comportent de la même façon que "dddd", en affichant le nom entier du jour. Ils n'affichent pas la date courte ou la date longue. **DayOfWeek** ("w") et **WeekOfYear** ("ww") ne sont pas prises en charge. À la place, vous pouvez utiliser la fonction **DatePart**, comme l'indique l'exemple suivant.

VB

```
    Format(DatePart(DateInterval.Weekday, Now))
...    Format(DatePart(DateInterval.WeekOfYear, Now))
```

"M" et "m" ne signifient pas la même chose. Par conséquent, tous deux respectent la casse. Utilisez "M" majuscule uniquement pour le mois dans la partie date d'un format de date/heure, et "m" minuscule uniquement pour les minutes de la partie heure.

Le spécificateur de format **Quarter** n'est pas pris en charge. À la place, vous pouvez utiliser la fonction **DatePart**, comme l'indique l'exemple suivant.

VB

```
Format(DatePart(DateInterval.Quarter, Now))
```

Pour afficher les minutes sous la forme d'un nombre avec ou sans zéros non significatifs, utilisez respectivement "mm" ou "m". Le format "ttttt" n'est plus pris en charge. "H" et "h" ne signifient pas la même chose. Par conséquent, tous deux respectent la casse. Utilisez "H" majuscule pour un système horaire sur 24 heures, "h" minuscule pour un système horaire sur 12 heures. Les formats AM/PM sont remplacés par "t" et "tt".

Le spécificateur "c" est utilisé pour la mise en forme des montants monétaires. Pour la mise en forme Date/Heure, utilisez "g" pour le spécificateur de date/heure courte et "G" pour le spécificateur de date/heure général. À la fois "g" et "G" utilisent vos paramètres régionaux actuels pour déterminer les formats de date et d'heure appropriés.

Formats numériques définis par l'utilisateur

Visual Basic 6.0

Dans Visual Basic 6.0, la fonction **Format** convertit des chaînes en nombres si nécessaire avant de les mettre en forme. **Format** affiche une virgule décimale de fin en l'absence de partie fractionnaire.

Visual Basic 6.0 prend en charge quatre sections dans la chaîne de mise en forme. Ceux-ci sont séparés par des points-virgules (**;**) et spécifient comment mettre en forme des valeurs positives, négatives, zéro et null. Les nombres négatifs comportant une section négative vide de la chaîne de format affichent une chaîne vide.

La mise en forme de notation scientifique prend en charge à la fois les espaces réservés aux chiffres "0" et "#"qui suivent l'exposant.

Visual Basic 2005

Dans Visual Basic 2005, la fonction **Format** ne convertit pas de chaînes en nombres avant la mise en forme. Vous devez passer un nombre dans le premier argument, et non une chaîne. Dans l'exemple suivant, la première ligne de code ne produit pas le résultat prévu dans Visual Basic 2005, alors que c'est le cas pour la deuxième ligne.

VB

```
    MsgBox(Format("1.234", "#.#"))   ' Displays "#.#".
...    MsgBox(Format(CSng("1.234"), "#.#"))   ' Displays "1.2".
```

Les virgules décimales de fin ne sont pas affichées, comme le montre l'exemple suivant :

VB

```
    MsgBox(Format(123, "###."))   ' Displays "123"
...    MsgBox(Format(123, "###.#"))   ' Displays "123"
```

Visual Basic 2005 prend en charge trois sections dans la chaîne de mise en forme, pour mettre en forme des valeurs positives, négatives et zéro. Si une valeur différente de zéro est arrondie à zéro suivant la première ou la deuxième section de format, elle est mise en forme suivant la troisième section. Les nombres négatifs comportant une section de la chaîne de format négative vide affichent un signe moins, comme dans l'exemple suivant.

VB

```
MsgBox(Format(-1, ";"))   ' Displays "-".
```

La mise en forme de notation scientifique prend uniquement en charge l'espace réservé au chiffre "0" et non "#". Dans l'exemple suivant, la première ligne de code ne produit pas le résultat prévu dans Visual Basic 2005, alors que c'est le cas pour la deuxième ligne.

VB

```
    MsgBox(Format(123, "#e+#"))   ' Displays "12e+3".
...    MsgBox(Format(123, "#e+0"))   ' Displays "1e+2".
```

Format de chaîne

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez créer des expressions pour les chaînes de format définies par l'utilisateur à l'aide des spécificateurs **@**, **&**, **<**, **>**, et **!**.

Visual Basic 2005

Visual Basic 2005 supprime la prise en charge des chaînes de format définies par l'utilisateur. Par conséquent, les spécificateurs de format **@**, **&**, **<**, **>** et **!** n'ont aucune signification et ne sont plus pris en charge.

Voir aussi

Référence

Format, fonction

DatePart, fonction (Visual Basic)

Concepts

Date et Time pour les utilisateurs de Visual Basic 6.0

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Date et Time pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 met à jour les fonctions **Date**, **Date$**, **Time** et **Time$** et remplace les fonctions **Now** et **Timer** par des propriétés.

Visual Basic 6.0

Dans Visual Basic 6.0, les fonctions **Date** et **Time** retournent la date et l'heure système au format **Date** codé sur quatre octets. De même, les instructions **Date** et **Time** utilisent le format Visual Basic 6.0 pour définir la date et l'heure système.

Les fonctions **Date$** et **Time$** retournent la date et l'heure système au format **String**.

Les fonctions **Now** et **Timer** spécifient la date et l'heure actuelles, ainsi que le nombre de secondes écoulées depuis minuit.

Visual Basic 2005

Visual Basic 2005 remplace **Date** et **Time** par les propriétés **Today** et **TimeOfDay** qui utilisent la structure DateTime codée sur huit octets du Common Language Runtime (CLR). Celle-ci correspond au type de données **Date** dans Visual Basic 2005. Vous pouvez utiliser **Today** et **TimeOfDay** à la fois pour définir et pour extraire la date et l'heure système.

Visual Basic 2005 remplace les fonctions **Date$** et **Time$** par les propriétés **DateString** et **TimeString**. Vous pouvez utiliser **DateString** et **TimeString** à la fois pour définir et pour extraire la date et l'heure système.

Les fonctions **Now** et **Timer** ont été remplacées par les propriétés en lecture seule de mêmes noms. Leurs séquences d'appel restent inchangées.

Voir aussi

Référence

Format, fonction
Today, propriété
TimeOfDay, propriété
Type de données date (Visual Basic)
DateString, propriété
TimeString, propriété
Now, propriété
Timer, propriété

Concepts

Fonction Format pour les utilisateurs de Visual Basic 6.0
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Fonction String ($) pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 évite que certaines fonctions aient besoin d'une version **String** et d'une version **Variant**.

Visual Basic 6.0

Certaines fonctions Visual Basic 6.0 possèdent deux versions, l'une pour retourner une valeur **String** et l'autre pour retourner une valeur **Variant**. Ces paires de fonctions se différencient par un suffixe dollar ($) pour la version **String**, par exemple **Chr** et **Chr$**.

Visual Basic 2005

Dans Visual Basic 2005, chaque paire de fonctions a été remplacée par une seule fonction. Les versions **Variant** sont supprimées, et vous pouvez appeler les versions **String** avec ou sans le suffixe $.

Même si seulement **Trim** est défini, Visual Basic 2005 accepte **Trim$**, car le suffixe **$** fonctionne comme un caractère de type d'identificateur pour le type de données **String**.

Voir aussi

Référence

Fonctions Trim, LTrim et Rtrim

Concepts

Fonction Format pour les utilisateurs de Visual Basic 6.0

Date et Time pour les utilisateurs de Visual Basic 6.0

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications diverses du langage pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux opérateurs et à la gestion de fichiers entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Opérateur booléen pour les utilisateurs de Visual Basic 6.0

Traite des opérateurs booléens et des expressions logiques.

Gestion des fichiers pour les utilisateurs de Visual Basic 6.0

Traite du développement des fonctionnalités de gestion de fichiers ainsi que de la compatibilité avec les précédentes versions des fonctions d'E/S de fichier de Visual Basic.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique d'éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Opérateur booléen pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, deux opérateurs booléens ont été supprimés et deux autres ajoutés, afin d'accroître les performances.

Visual Basic 6.0

Dans Visual Basic 6.0, les opérateurs booléens (**And**, **Or**, **Not** et **Xor**) continuent à évaluer toutes les expressions faisant partie de leurs opérandes.

Les opérateurs **Eqv** et **Imp** permettent d'effectuer des équivalences logiques et des implications entre deux expressions.

Visual Basic 2005

Dans Visual Basic 2005, les opérateurs **And**, **Or**, **Not** et **Xor** continuent d'évaluer toutes les expressions contribuant à leurs opérandes. Dans Visual Basic 2005 sont également introduits deux nouveaux opérateurs, **AndAlso** et **OrElse**, qui peuvent réduire le temps d'exécution en *court-circuitant* des évaluations logiques. Si le premier opérande d'un opérateur **AndAlso** a la valeur **False**, le deuxième n'est pas évalué. De même, si le premier opérande d'un opérateur **OrElse** a la valeur **True**, le deuxième n'est pas évalué.

Remarque

Soyez prudent lorsque vous utilisez des opérateurs court-circuitant. Si le deuxième opérande contient un appel de procédure, vous n'êtes pas certain que cette procédure sera appelée à chaque exécution de l'opérateur.

Les opérateurs **Eqv** et **Imp** ne sont pas pris en charge. Utilisez l'opérateur de comparaison égal (**=**) à la place de **Eqv** pour les évaluations logiques et de bits. Vous pouvez remplacer l'opérateur logique **Imp** par une expression avec les opérateurs **Not** et **Or**, comme le montre l'exemple suivant :

VB

```
Result = (Not A) Or B  'Same as A Imp B
                       '(True unless A is True and B is False.)
```

Vous pouvez remplacer un opérateur de bits **Imp** de la même manière, en appliquant des opérateurs **Not** et **Or** à des opérandes numériques.

Voir aussi

Référence

And, opérateur (Visual Basic)
AndAlso, opérateur
Or, opérateur (Visual Basic)
OrElse, opérateur
Not, opérateur (Visual Basic)
Xor, opérateur (Visual Basic)
TimeSpan

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Gestion des fichiers pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 offre plus de fonctionnalités de gestion de fichiers ainsi que la compatibilité avec les précédentes versions des fonctions d'E/S de fichier de Visual Basic.

Visual Basic 6.0

Dans Visual Basic 6.0, la gestion de fichiers utilise différentes fonctions d'E/S de fichier telles que **Open**, **Input**, **Output** et **Append**. L'objet **FileSystemObject** fournit une méthode orientée objet d'utilisation des fichiers.

Visual Basic 2005

Dans Visual Basic 2005, la gestion de fichiers s'effectue à l'aide de l'objet My.Computer.FileSystem. En outre, la classe **System.IO.File** offre des fonctions qui sont compatibles avec les anciennes fonctions d'E/S de fichier Visual Basic.

La classe FileStream permet d'accéder à des fichiers d'entrée et de sortie standard, ainsi qu'à des périphériques d'erreur standard.

Le tableau suivant répertorie les membres de l'objet **My.Computer.FileSystem** disponibles dans Visual Basic 2005.

Membre	Description
CombinePath	Retourne un chemin d'accès combiné correctement mis en forme sous la forme d'une **String**.
CopyDirectory	Copie un répertoire.
CopyFile	Copie un fichier.
CurrentDirectory	Obtient ou définit le répertoire actif.
CreateDirectory	Crée un répertoire.
DeleteDirectory	Supprime un répertoire.
DeleteFile	Supprime un fichier.
DirectoryExists	Retourne une **Boolean** qui indique si un répertoire existe.
Lecteurs	Retourne une collection en lecture seule répertoriant tous les noms de lecteurs disponibles.
FileExists	Retourne une **Boolean** qui indique si un fichier existe.
FindInFiles	Retourne une collection en lecture seule de chaînes qui représentent les noms des fichiers contenant le te xte spécifié.
GetDirectories	Retourne une collection **String** représentant les chemins d'accès des sous-répertoires présents dans un r épertoire défini.
GetDirectoryInfo	Retourne un objet DirectoryInfo pour le chemin d'accès spécifié.
GetDriveInfo	Retourne un objet DriveInfo pour le chemin d'accès spécifié.
GetFileInfo	Retourne un objet FileInfo pour le chemin d'accès spécifié.
GetFiles	Retourne une collection **String** en lecture seule qui représente les noms des fichiers d'un répertoire.
GetParentPath	Retourne une **String** qui représente le chemin d'accès absolu du parent du chemin d'accès spécifié.

GetTempFileName	Crée un fichier temporaire de zéro octet nommé de manière unique sur le disque et retourne le chemin d 'accès complet à ce fichier.
MoveDirectory	Déplace un répertoire à l'emplacement spécifié.
MoveFile	Déplace un fichier à l'emplacement spécifié.
OpenTextFieldParser	Ouvre un TextFieldParser.
OpenTextFileReader	Ouvre un TextReader.
OpenTextFileWriter	Ouvre un TextWriter.
ReadAllBytes	Lit à partir d'un fichier.
ReadAllText	Lit à partir d'un fichier texte.
RenameDirectory	Renomme un répertoire.
RenameFile	Renomme un fichier.
SpecialDirectories	Obtient un objet qui fournit les propriétés pour l'accès aux répertoires communément référencés
WriteAllBytes	Écrit dans un fichier binaire.
WriteAllText	Écrit dans un fichier texte.

Voir aussi

Référence

System.IO Namespace
My.Computer.FileSystem, objet
My.Computer.FileSystem.SpecialDirectories, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser
Codages de fichiers
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic
Création, suppression et déplacement de fichiers et de répertoires en Visual Basic
Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Modifications des modules pour les utilisateurs de Visual Basic 6.0

Une modification de syntaxe affecte la déclaration de modules privés.

Instruction Option Private Module

Visual Basic 6.0

Dans Visual Basic 6.0, l'instruction **Option Private Module** indique que le module tout entier est privé et que ses composants ne sont pas accessibles à l'extérieur de ce projet.

Visual Basic 2005

Visual Basic 2005 ne prend pas en charge l'instruction **Option Private Module**. Vous ne pouvez pas déclarer un module comme privé. Toutefois, vous pouvez déclarer tous ou aucun de ses membres comme privé.

Contrairement aux modules, les classes peuvent être privées. Si vous avez besoin des fonctionnalités du module qui est lui-même privé, déclarez une classe à la place.

Voir aussi

Référence

Module, instruction
Class, instruction (Visual Basic)
Private (Visual Basic)

Modifications des procédures pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux procédures entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

Explique comment passer des arguments aux procédures.

Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0

Traite des appels de procédure et des retours.

Déclaration de procédure pour les utilisateurs de Visual Basic 6.0

Décrit les arguments de procédure optionnels et les variables locales statiques.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique des éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 introduit plusieurs modifications expliquant comment les arguments sont passés aux procédures.

Mécanisme de passage d'arguments par défaut

Visual Basic 6.0

Dans Visual Basic 6.0, si vous ne spécifiez pas **ByVal** ou **ByRef** dans un paramètre de procédure, le mécanisme de passage utilise **ByRef** par défaut. De cette façon, la variable passée dans la procédure peut être modifiée dans le programme appelant.

Exposer une variable à la modification peut comporter certains dangers. Dans l'exemple suivant, le mécanisme de passage utilise par défaut **ByRef**, la valeur `elapsedSeconds` est modifiée par `minutesPastHour`, et `elapsedSeconds` est donc affichée de façon incorrecte par **MsgBox** :

```
Function minutesPastHour(seconds As Integer) As Integer
    Dim hours As Integer = seconds \ 3600
    seconds = seconds Mod 3600
    Return seconds \ 60
End Function
Sub showSecondsAndMinutes()
    Dim elapsedSeconds, extraMinutes As Integer
    elapsedSeconds = CInt(Timer()) ' Integer seconds since midnight.
    extraMinutes = minutesPastHour(elapsedSeconds)
    MsgBox "Total seconds: " & elapsedSeconds & _
        "; minutes past hour: " & extraMinutes
End Sub
```

Le passage d'un argument **ByRef** permet à une procédure de modifier cet argument dans le programme appelant, ce qui peut donner lieu à un comportement imprévu. De plus, si cette procédure en appelle une autre et passe le même argument **ByRef**, les risques de voir la variable d'origine modifiée par inadvertance sont accrus.

Visual Basic 2005

Dans Visual Basic 2005, le mécanisme de passage utilise par défaut **ByVal** pour tous les paramètres lors de la déclaration d'une procédure. Il s'agit là d'une mesure destinée à empêcher toute modification des arguments. La déclaration de l'exemple précédent peut être réécrite comme suit.

VB

```
Function MinutesPastHour(ByVal Seconds As Integer) As Integer
```

Le passage de `seconds` par valeur empêche que la procédure n'accède à la variable dans le programme appelant et contribue à éviter les dangers que nous venons de décrire.

Bien que **ByVal** soit utilisé comme mécanisme de passage par défaut, en le spécifiant explicitement pour chaque paramètre, le code, libéré de toute ambiguïté, devient plus facile à lire.

Arguments de propriété ByRef

Visual Basic 6.0

Dans Visual Basic 6.0, une propriété passée à une procédure en tant qu'argument **ByRef** est copiée dans cette procédure, mais elle ne peut pas en être extraite. Cela signifie que toute modification d'un argument de propriété de ce genre n'est pas réfléchie sur la propriété d'origine dans le programme appelant, bien que cet argument ait été passé avec **ByRef**.

Visual Basic 2005

Dans Visual Basic 2005, un argument de propriété passé **ByRef** est copié à la fois en dedans et en dehors de la procédure. L'exemple suivant montre comment une propriété peut être modifiée par une procédure.

VB

```
Sub Reduce(ByRef Height As Single)
' ... ... ... ... ... ... ...  ' Code to modify Height argument.
```

```
End Sub
```

VB

```
Dim Sq As Square = New Square  ' Square has property Side of type Single.
Reduce(Sq.Side)                ' Side is changed when Reduce returns.
```

Quand une procédure modifie un argument de propriété, la valeur de la propriété d'origine n'est pas modifiée immédiatement dans le programme appelant. Au lieu de cela, elle est extraite dès le retour à la procédure.

Arguments ParamArray

Visual Basic 6.0

Dans Visual Basic 6.0, une procédure peut spécifier le mot clé **ParamArray** sur le dernier de ses arguments pour accepter un tableau d'arguments **Variant**. Il est impossible de déclarer le mécanisme de passage de ces arguments. Ces derniers sont toujours passés avec **ByRef**.

Visual Basic 2005

Dans Visual Basic 2005, les arguments **ParamArray** sont toujours passés **ByVal**. Les arguments du tableau doivent tous être du même type de données que l'argument **ParamArray**.

Voir aussi

Référence

Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
ByVal
ByRef
ParamArray

Concepts

Déclaration de procédure pour les utilisateurs de Visual Basic 6.0
Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0
Tableaux de paramètres
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, plusieurs changements ont été introduits sur le plan des séquences d'appel des procédures. Ces modifications améliorent la cohérence de la syntaxe.

Parenthèses dans les appels de procédure

Visual Basic 6.0

Dans Visual Basic 6.0, la présence de parenthèses est obligatoire autour de la liste d'arguments dans les appels **Function**. Dans les appels **Sub**, elles ne sont requises que si vous utilisez l'instruction **Call** et interdites si ce n'est pas le cas. L'exemple suivant montre une instruction appelante valide :

```
y = Sqrt(x)

Call displayCell(2, 14, cellValue)

displayCell 2, 14, cellValue ' Variation on previous statement.
```

Visual Basic 2005

Dans Visual Basic 2005, la présence de parenthèses autour d'une liste d'arguments non vide est toujours requise dans n'importe quel appel de procédure. Dans les appels **Sub**, l'instruction **Call** est facultative. L'exemple précédent peut être réécrit de la manière suivante :

VB

```
    Y = Math.Sqrt(X)
...    DisplayCell(2, 14, Value)
```

Si vous appelez une procédure sans fournir d'argument, vous pouvez utiliser des parenthèses vides ou ne pas en utiliser du tout.

Return, instruction

Visual Basic 6.0

Dans Visual Basic 6.0, l'instruction **Return** peut être utilisée uniquement pour rebrancher le code après une instruction **GoSub**. Les deux instructions doivent figurer dans la même procédure.

Visual Basic 2005

Dans Visual Basic 2005, l'instruction **GoSub** n'est pas prise en charge, et vous pouvez recourir à l'instruction **Return** pour retourner le contrôle au programme appelant à partir d'une procédure **Function** ou **Sub**.

Voir aussi

Référence

Return, instruction (Visual Basic)
Call, instruction (Visual Basic)

Concepts

Syntaxe de déclaration pour les utilisateurs de Visual Basic 6.0
Déclaration de procédure pour les utilisateurs de Visual Basic 6.0
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0
Instruction de contrôle pour les utilisateurs de Visual Basic 6.0
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Déclaration de procédure pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 introduit plusieurs modifications liées à la déclaration des procédures.

Arguments facultatifs

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez déclarer un paramètre de procédure comme **Optional** sans spécifier de valeur par défaut. Si un paramètre facultatif est de type **Variant**, le code de procédure peut utiliser la fonction **IsMissing** pour déterminer si le code appelant a passé un argument à ce paramètre.

Visual Basic 2005

Dans Visual Basic 2005, tous les paramètres facultatifs doivent déclarer une valeur par défaut qui est passée à la procédure si le programme appelant ne fournit pas cet argument. La fonction **IsMissing** n'est pas nécessaire pour détecter un argument manquant et n'est pas prise en charge. L'exemple suivant montre la déclaration d'un argument optionnel :

VB

```
Sub Calculate(Optional ByVal Switch As Boolean = False)
```

Variables locales statiques

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez déclarer une procédure avec le modificateur **Static**. De cette façon, toutes les variables locales de la procédure deviennent statiques et conservent leur valeur entre les appels.

Visual Basic 2005

Dans Visual Basic 2005, le mot clé **Static** n'est pas pris en charge dans une instruction **Function** ou **Sub**. Vous devez déclarer individuellement chaque variable locale voulue comme **Static**.

Voir aussi

Référence

Optional (Visual Basic)
Static (Visual Basic)

Concepts

Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications des propriétés pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées aux propriétés entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0

Traite des changements apportés en termes de prise en charge des propriétés par défaut.

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

Traite des modifications apportées en termes d'arguments et de procédures de propriété.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique d'éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, la prise en charge des propriétés par défaut a été modifiée en vue d'une simplification et d'une amélioration de la lisibilité.

Visual Basic 6.0

Dans Visual Basic 6.0, les propriétés par défaut sont prises en charge sur les objets. Prenons par exemple un contrôle **Label** ayant pour propriété par défaut **Caption**. Les deux assignations de l'exemple suivant sont équivalentes.

```
Dim lbl As Label
lbl = "Important"
lbl.Caption = "Important"
```

Si les propriétés par défaut vous permettent d'abréger quelque peu le code Visual Basic lors de son écriture, elles présentent plusieurs inconvénients :

- Elles peuvent entraver la lisibilité du code. Dans l'exemple précédent, si vous ne connaissez pas bien le contrôle **Label**, la première assignation ne suffit pas à déterminer si la chaîne `"Important"` est stockée directement dans la variable `lbl` ou dans une propriété par défaut.
- En partant d'un objet que vous comptez utiliser dans le code, il n'est pas toujours aisé de déterminer s'il possède une propriété par défaut et, même si c'est le cas, de quelle propriété il s'agit.
- Les propriétés par défaut rendent obligatoire la présence de l'instruction **Set** dans le langage Visual Basic. L'exemple suivant démontre que **Set** est nécessaire pour indiquer que c'est une référence d'objet, et non une propriété par défaut, qui doit être assignée.

  ```
  Dim lbl1 As Label, lbl2 As Label
  lbl1 = "Saving" ' Assign a value to lbl1's Caption property.
  lbl2 = lbl1       ' Replace lbl2's Caption property with lbl1's.
  Set lbl2 = lbl1   ' Replace lbl2 with an object reference to lbl1.
  ```

Visual Basic 2005

Dans Visual Basic 2005, les propriétés par défaut ne sont pas prises en charge à moins d'accepter des arguments. En raison de cette modification syntaxique, les instructions **Let** et **Set** ne sont pas nécessaires pour spécifier ce qu'il faut assigner et elles ne sont pas employées dans les instructions d'assignation. La propriété Text remplace la propriété **Caption** du contrôle Label, et l'exemple précédent peut être réécrit comme suit.

VB

```
Dim L1, L2 As New Label   ' Both become type Label
                          ' in the new version of Visual Basic.
L1.Text = "Saving"        ' Assign Text property.
L2.Text = L1.Text         ' Copy Text property.
L2 = L1                   ' Copy object reference.
```

Let est encore un mot réservé dans Visual Basic 2005, bien qu'il ne soit pas utilisé d'un point de vue syntaxique. Cela permet d'éviter la confusion avec ses significations précédentes. Visual Basic 2005 utilise l'instruction **Set** pour les procédures de propriété qui définissent la valeur d'une propriété.

Propriétés paramétrées

Les propriétés par défaut qui acceptent des arguments ne sont pas ambiguës et sont prises en charge dans Visual Basic 2005. Les propriétés par défaut sont le plus souvent associées à des classes de collection. Par exemple, dans l'espace de noms System.Windows.Forms, la classe Form prend en charge la hiérarchie suivante :

Objet **Form**

Propriété Controls (retourne un objet ControlCollection de ce formulaire)

Objet **ControlCollection** (la propriété par défaut est Item)

Propriété **Item** (retourne un objet Control pour un élément de la collection)

Objet **Control**

La propriété **Controls** retourne un objet **ControlCollection** et la propriété **Item** retourne un objet **Control**. L'exemple suivant illustre des cas d'utilisation valide et non valide de propriétés par défaut dans Visual Basic 2005.

VB

```
Dim F As New Form   ' Assume F has been created and initialized.
```

VB

```
F.Controls.Item(0).Text = "Stop"  ' Valid -- no default properties used.
F.Controls(0).Text = "Stop"  ' Valid -- Item is parameterized.
```

VB

```
'F(0).Text = "Stop"  ' INVALID -- Form does not have a default property.
'F.Controls(0) = "Stop"  ' INVALID -- No default property on Control.
```

Déclaration de propriété par défaut

Dans Visual Basic 2005, pour spécifier une propriété comme la propriété par défaut, vous devez faire débuter sa déclaration par le mot clé **Default**. Si vous surchargez le nom de la propriété, vous devez spécifier **Default** dans toutes les déclarations de surcharge. Vous ne pouvez pas déclarer une propriété par défaut comme **Shared** ou **Private**.

Voir aussi

Référence

Set, instruction (Visual Basic)
Default (Visual Basic)
Text
Label
System.Windows.Forms
Form
ControlCollection
Control

Concepts

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, des mises à jour ont été apportées concernant la déclaration des procédures de propriété et des paramètres à des fins de simplification et en vue d'assurer l'interopérabilité avec d'autres langages de programmation.

Visual Basic 6.0

Dans Visual Basic 6.0, utilisez les instructions **Property Get**, **Property Let** et **Property Set** pour obtenir et définir des valeurs de propriété.

Vous pouvez déclarer un paramètre de propriété comme étant **ByRef**. Le passage d'une variable à un tel paramètre permet une procédure de modification de cette variable dans le code appelant.

Visual Basic 2005

Visual Basic 2005 introduit une syntaxe de déclaration de propriété unifiée qui englobe les procédures destinées à obtenir et à définir la valeur de la propriété. Cette syntaxe garantit la cohérence d'attributs de propriété tels que le niveau d'accès et la surcharge. L'exemple suivant illustre la déclaration d'une propriété qui ne prend pas de paramètres :

VB

```
Private monthNum As Integer = 1
Property month() As Integer
    Get
        Return monthNum
    End Get
    Set(ByVal Value As Integer)
        If Value < 1 Or Value > 12 Then
            ' Error processing for invalid value.
        Else
            monthNum = Value
        End If
    End Set
End Property
```

Vous pouvez utiliser le nom de votre choix pour le paramètre **Set**. Si vous ne fournissez pas d'argument, `Value` est automatiquement généré.

Cette modification de syntaxe rend les instructions **Property Get Property Set** inutiles, ainsi elles ne sont pas prises en charge. Comme Visual Basic 2005 n'autorise pas de propriétés par défaut sans paramètres, **Property Let** et **Property Set** ne sont pas obligatoires pour distinguer s'il faut assigner une référence d'objet ou une propriété par défaut. Par conséquent, l'instruction **Property Let** n'est pas prise en charge.

Visual Basic 2005 ne prend pas en charge les paramètres de la propriété **ByRef**. Si une procédure de propriété possédait un paramètre **ByRef** et si elle modifiait la variable sous-jacente de l'argument correspondant, la propriété peut se comporter de façon inattendue. Par conséquent, toutes les déclarations de propriétés paramétrées doivent spécifier **ByVal** pour les arguments.

Voir aussi

Référence

ByVal

Concepts

Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Modifications du flux de contrôle pour les utilisateurs de Visual Basic 6.0

Les rubriques suivantes décrivent les modifications apportées au flux de contrôle entre Visual Basic 6.0 et Visual Basic 2005.

Dans cette section

Instruction de contrôle pour les utilisateurs de Visual Basic 6.0

Traite des modifications apportées aux instructions de contrôle du flux d'exécution.

Gestion des exceptions pour les utilisateurs de Visual Basic 6.0

Traite de la prise en charge de la gestion structurée des exceptions, et de la poursuite de la prise en charge de la gestion non structurée des exceptions.

Rubriques connexes

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Propose une liste alphabétique des éléments de programmation qui ont changé ou qui ne sont plus pris en charge dans Visual Basic 2005 et indique les éléments de remplacement.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles et modifiées de Visual Basic 2005.

Instruction de contrôle pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, plusieurs instructions de contrôle du flux d'exécution ont été soumises à des modifications.

Visual Basic 6.0

Dans Visual Basic 6.0, l'instruction **GoSub** appelle une sous-procédure d'une procédure. Les constructions **On ... GoSub** et **On ... GoTo**, également appelées **GoSub** calculée et **GoTo** calculée, assurent la compatibilité avec les anciennes versions de BASIC.

La construction **While ... Wend** effectue une boucle dans le code tant qu'une condition spécifiée est true.

Visual Basic 2005

Dans Visual Basic 2005, vous pouvez appeler des procédures avec l'instruction **Call** et l'instruction **GoSub** n'est pas prise en charge. Vous pouvez exécuter plusieurs branchements avec les instructions **Select...Case** et les constructions **On ... GoSub** et **On ... GoTo** ne sont pas prises en charge. Toutefois, Visual Basic 2005 prend encore en charge l'instruction **On Error**.

Visual Basic 2005 conserve la construction **While ... Wend**, mais il remplace le mot clé **Wend** par l'instruction **End**. Le mot clé **Wend** n'est pas pris en charge.

Voir aussi

Référence

Call, instruction (Visual Basic)
Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Return, instruction (Visual Basic)
Select...Case, instruction (Visual Basic)
On Error, instruction (Visual Basic)
End, instruction

Concepts

Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Gestion des exceptions pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 ajoute la prise en charge de la gestion structurée des exceptions, et continue de prendre en charge la gestion non structurée des exceptions.

Visual Basic 6.0

Dans Visual Basic 6.0, vous utilisez la gestion non structurée des exceptions pour traiter les erreurs dans votre code. Le fait de placer l'instruction **On Error** au début d'un bloc de code gère toutes les erreurs qui se produisent dans ce bloc. La gestion non structurée des exceptions utilise également les instructions **Error** et **Resume**.

Visual Basic 2005

Dans Visual Basic 2005, le code de gestion structurée des exceptions détecte et répond aux erreurs d'exécution en associant une structure de contrôle à des exceptions, des blocs de code protégés et des filtres. La gestion structurée des exceptions est gérée par une instruction **Try** qui est composée de trois types de blocs : **Try**, **Catch** et **Finally**. Un bloc **Try** est un bloc d'instructions contenant l'instruction à exécuter. Un bloc **Catch** est un bloc d'instructions qui gère une exception. Un bloc **Finally** contient des instructions à exécuter lorsque l'instruction **Try** se termine, qu'une exception se soit produite ou non. L'instruction **Throw**, utilisée en association avec un bloc **Catch**, génère une exception qui est représentée par une instance d'un type dérivé de la classe Exception.

Voir aussi

Référence

Throw, instruction (Visual Basic)
On Error, instruction (Visual Basic)
Error, instruction
Resume, instruction
Exception

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic
Vue d'ensemble de la gestion non structurée des exceptions
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Programmation orientée objet pour les utilisateurs de Visual Basic 6.0

Visual Basic 6.0 prend en charge des éléments de langage orientés objet, ainsi que les objets distribués dans les bibliothèques. Visual Basic 2005 étend la prise en charge de la programmation orientée objet en prenant en charge toutes ses propriétés de langage.

Différences conceptuelles

Les sections suivantes répertorient les fonctionnalités orientées objet dans Visual Basic 2005, en les comparant avec l'implémentation Visual Basic 6.0 quand cela est possible. Pour chaque fonctionnalité, des liens sont fournis vers des pages d'aide complètes qui vous permettront d'en apprendre plus sur le sujet.

Niveaux d'accès

Dans Visual Basic 6.0, les mots clés **Private**, **Friend**, **Public** et **Static** sont utilisés pour définir des niveaux d'accès pour les éléments déclarés.

Dans Visual Basic 2005, les mots clés **Private**, **Friend**, **Public**et **Static**, plus les nouveaux mots clés **Protected** et **Protected Friend** sont utilisés pour définir des niveaux d'accès pour les éléments déclarés. Le niveau d'accès d'un élément déclaré est l'étendue de la capacité à l'utiliser, à savoir le code qui est autorisé à lire l'élément ou à y écrire.

Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic et Portée dans Visual Basic.

Attributs

Dans Visual Basic 6.0, une prise en charge limitée des attributs incorporés est fournie par des outils tels que les attributs de procédure dans l'IDE Visual Basic.

Dans Visual Basic 2005, un **Attribute** est une balise descriptive qui peut être utilisée pour annoter des instructions de type et de membre de type, en modifiant ainsi leur signification ou en personnalisant leur comportement. Par exemple, des instructions de classe et des instructions de méthode de classe peuvent être annotées avec des attributs.

Votre application et d'autres applications, telles que le compilateur Visual Basic, peuvent utiliser la réflexion pour accéder à des attributs afin de déterminer comment des types et des membres de type peuvent être utilisés.

Les attributs peuvent être utilisés pour effectuer la programmation AOP (Aspect Oriented Programming) avec Visual Basic. Un aspect est une partie d'un programme qui traverse sa logique métier. En d'autres termes, il est nécessaire pour exécuter le programme, mais n'est pas nécessairement spécifique au domaine pour lequel le programme est écrit. Isoler des aspects tels que l'enregistrement et la persistance de la logique métier est l'objectif du paradigme de programmation orienté aspect.

Pour plus d'informations, consultez Vue d'ensemble des attributs dans Visual Basic, Attributs utilisés dans Visual Basic et Utilisations courantes des attributs.

Compatibilité binaire

Dans Visual Basic 6.0, l'option **Binary Compatibility** vous permet de conserver automatiquement les identificateurs de classe et d'interface à partir d'une version antérieure d'un composant lors de la compilation d'une nouvelle version.

Dans Visual Basic 2005, l'option **Binary Compatibility** n'est pas prise en charge. À la place, la compatibilité binaire est effectuée avec des attributs. Ceci vous donne un contrôle direct sur les informations situées dans votre composant compilé, telles que les identificateurs de classe et d'interface, les offsets de table virtuelle et les attributs COM appropriés.

Pour plus d'informations, consultez Modifications de la compatibilité binaire pour les utilisateurs de Visual Basic 6.0.

Modules de classe

Dans Visual Basic 6.0, une classe est définie dans un module de classe. Un seul module de classe est stocké dans un type spécial de fichier qui possède une extension de fichier .cls.

Dans Visual Basic 2005, une classe est définie dans une instruction **Class** qui spécifie le nom et les membres d'une classe. Les instructions **Class** sont stockées dans des fichiers sources. Le fichier source entier peut être affiché comme du texte brut.

Plusieurs instructions **Class** ainsi que d'autres instructions de type peuvent être stockées dans un seul fichier source. Visual Basic ne nécessite pas que le nom du fichier source corresponde à une **Class** ou un type défini dans le fichier source.

Pour plus d'informations, consultez Class, instruction (Visual Basic).

Méthodes de constructeur de classe

Dans Visual Basic 6.0, le gestionnaire d'événements **Initialize** de la classe, nommé **Class_Initialize**, est utilisé pour exécuter le code qui doit être exécuté au moment où un objet est créé.

Dans Visual Basic 2005, un ou plusieurs constructeurs sont ajoutés dans une classe pour exécuter le code et initialiser des variables. Les constructeurs sont les méthodes dans une classe qui sont nommées **New**. La méthode **New** peut être surchargée pour fournir à plusieurs constructeurs le nom **New** dans la même instruction de classe.

Pour plus d'informations, consultez New (Visual Basic) ou Utilisation de constructeurs et de destructeurs.

Méthodes de destructeur de classe

Dans Visual Basic 6.0, le mot clé **Nothing** est utilisé pour dissocier une variable d'objet d'un objet réel. L'instruction **Set** est utilisée pour assigner **Nothing** à une variable d'objet.

Dans Visual Basic 2005, pour la majorité des objets créés par votre application, vous pouvez laisser au garbage collector le soin de réaliser automatiquement les tâches de gestion de mémoire requises. Cependant, les ressources non managées requièrent un nettoyage explicite. Le type le plus répandu de ressource non managée est un objet qui encapsule une ressource de système d'exploitation telle qu'un handle de fichier ou de fenêtre ou une connexion réseau.

Lors de la création d'un objet qui encapsule une ressource non managée, il est recommandé de fournir le code nécessaire pour nettoyer la ressource non managée dans une méthode `Dispose` publique. En fournissant une méthode `Dispose`, vous donnez la possibilité aux utilisateurs de votre objet d'en libérer explicitement la mémoire lorsqu'ils ont fini de s'en servir.

Lorsque vous utilisez un objet qui encapsule une ressource non managée, vous devez garder à l'esprit la méthode `Dispose` et l'appeler si nécessaire.

Pour plus d'informations, consultez Nettoyage de ressources non managées ou Gestion automatique de la mémoire.

Class_Initialize

Dans Visual Basic 6.0, un événement **Initialize** de classe peut être traité par une méthode **Class_Initialize** pour exécuter le code devant être exécuté au moment où un objet est créé. Par exemple, les valeurs des variables de données de classe peuvent être initialisées.

Dans Visual Basic 2005, l'événement **Initialize** et le gestionnaire **Class_Initialize** ne sont pas pris en charge. Pour fournir l'initialisation de classe, ajoutez une ou plusieurs méthodes de constructeur aux classes et aux structures que vous définissez.

Pour plus d'informations, consultez Modifications de Class_Initialize pour les utilisateurs de Visual Basic 6.0.

Données, classes

Dans Visual Basic 6.0, des classes source de données et consommateur de données complexes sont utilisées pour fonctionner avec des données externes telles que des bases de données Microsoft SQL Server. Une classe source de données fournit des données provenant d'une source externe. Une classe consommateur de données peut être liée à une source de données externe comme une classe **Data Source**.

Dans Visual Basic 2005, des classes source de données, consommateur de données simples, consommateur de données complexes et des classes de liaison sont utilisées pour fonctionner avec des données externes et internes.

Pour plus d'informations, consultez Liaison de données et Windows Forms.

Default, propriété

Dans Visual Basic 6.0, toute propriété de classe peut être définie comme propriété par défaut de la classe.

Dans Visual Basic 2005, le membre par défaut d'une classe ou d'une structure peut uniquement être une propriété qui utilise un ou plusieurs arguments. Les membres de propriété par défaut sont définis en incluant le mot clé **Default** dans une instruction de déclaration de propriété dans une classe ou une structure.

Pour plus d'informations, consultez Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0.

Délégués

Dans Visual Basic 6.0, les types délégués ne sont pas pris en charge.

Dans Visual Basic 2005, un type délégué est une forme de pointeur méthode orienté objet permettant à une méthode d'être appelée indirectement par l'intermédiaire d'une référence à la méthode. Les délégués peuvent être utilisés pour raccorder des

gestionnaires d'événements et passer une méthode d'une méthode à une autre.

Les délégués peuvent être utilisés pour implémenter le modèle de design asynchrone. Par exemple, avec un délégué asynchrone, un programme peut appeler une méthode, qui énumère une longue liste, tandis que le programme principal poursuit son exécution. Lorsque l'énumération est terminée, un rappel est effectué et le programme lui répond.

Pour plus d'informations, consultez Délégués en Visual Basic ou
Comment : passer des procédures à une autre procédure dans Visual Basic.

Gestion des erreurs

Dans Visual Basic 6.0, l'instruction **On Error** est utilisée pour la gestion non structurée des exceptions.

Dans Visual Basic 2005, la gestion non structurée et la gestion structurée des exceptions sont prises en charge. La gestion structurée des exceptions désigne la création d'un mécanisme de gestion des exceptions à l'aide d'une structure de contrôle contenant des exceptions, des blocs de code isolés et des filtres.

Pour plus d'informations, consultez Gestion structurée des exceptions en Visual Basic ou
Cas d'utilisation de la gestion structurée ou non structurée des exceptions.

Événements

Dans Visual Basic 6.0, les mots clés **Event**, **RaiseEvent** et **WithEvents** sont utilisés pour déclarer, déclencher et gérer des événements.

Dans Visual Basic 2005, les mots clés **Event**, **RaiseEvent** et **WithEvents** et les nouveaux mots clés **AddHandler**, **RemoveHandler** et **Handles** sont utilisés pour déclarer, déclencher et gérer des événements.

Les mots clés **AddHandler** et **RemoveHandler** vous permettent d'ajouter, de supprimer et de changer dynamiquement le gestionnaire d'événements associé à un événement.

Le mot clé **Handles** vous permet de définir une clause **Handles** sur une méthode. **Handles** déclare qu'une procédure gère un événement spécifié.

Dans le .NET Framework, les événements reposent sur le modèle délégué. Le modèle délégué est basé sur le modèle de design observateur orienté objet.

Pour plus d'informations, consultez Comment : écrire des gestionnaires d'événements.

Génériques

Dans Visual Basic 6.0, les types génériques ne sont pas pris en charge.

Dans Visual Basic 2005, les génériques sont utilisés pour implémenter le polymorphisme paramétrique dans un programme Visual Basic. Le code générique est écrit sans mention de quelque type spécifique que ce soit et peut donc être utilisé de façon transparente avec n'importe quel nombre de nouveaux types.

Visual Basic 2005 prend en charge les paramètres de type sur les classes, structures, interfaces, procédures et délégués génériques. Un argument de type correspondant spécifie, au moment de compilation, le type de données de l'un des éléments dans le type générique.

Pour plus d'informations, consultez Types génériques en Visual Basic.

Classes globales

Dans Visual Basic 6.0, la valeur de la propriété **Instancing** d'une classe détermine si une classe est privée - c'est-à-dire réservée à une utilisation dans un seul composant - ou disponible pour d'autres applications. Il détermine également comment d'autres applications créent des instances de la classe et comment d'autres applications appellent les membres de classe.

Dans Visual Basic 2005, la propriété **Instancing** n'est plus prise en charge.

Vous pouvez appliquer le mot clé **Public** aux instructions de classe dans un assembly pour exposer les classes de cet assembly à d'autres assemblys.

Vous pouvez référencer un assembly externe tel qu'une bibliothèque de classes pour permettre au code dans votre application d'utiliser les classes **Public** de cette bibliothèque de classes.

Vous pouvez utiliser le mot clé **Imports** pour importer des noms d'espaces de noms provenant de projets et d'assemblys référencés. Le mot clé **Imports** peut également importer des noms d'espaces de noms définis dans le même projet que le fichier dans lequel figure l'instruction.

Vous pouvez appliquer le mot clé **Shared** aux membres de champ, de propriété et de méthode des classes et structures pour implémenter des membres partagés. Les membres partagés sont des propriétés, des procédures et des champs qui sont partagés par toutes les instances d'une classe ou structure. Les membres partagés d'une classe sont accessibles sans instanciation de la classe.

Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic, Références et l'instruction Imports ou Membres partagés en Visual Basic.

Héritage d'implémentation

Dans Visual Basic 6.0, l'héritage d'implémentation n'est pas pris en charge.

Dans Visual Basic 2005, vous pouvez implémenter le polymorphisme ad hoc via l'héritage d'implémentation. Cela vous permet de définir des classes pouvant être utilisées de manière interchangeable par code client au moment de l'exécution, mais avec des méthodes ou des propriétés fonctionnellement différentes (même si elles portent le même nom).

Vous pouvez définir des classes qui servent de base aux classes dérivées. Les classes dérivées héritent des propriétés, des méthodes et des événements de la classe de base et peuvent les étendre. Elles peuvent également substituer des méthodes héritées par de nouvelles implémentations.

Pour plus d'informations, consultez Éléments fondamentaux de l'héritage ou Quand utiliser l'héritage.

Héritage de l'interface

Visual Basic 6.0 assure le polymorphisme via plusieurs interfaces ActiveX. Dans le modèle COM (Component Object Model) qui forme l'infrastructure de la spécification ActiveX, plusieurs interfaces permettent aux systèmes de composants logiciels d'évoluer sans rupture du code existant.

Dans Visual Basic 2005, vous pouvez implémenter le polymorphisme ad hoc avec le mot clé **Interface** du .NET Framework.

Plusieurs classes peuvent implémenter la même **Interface**, et une seule classe peut implémenter une ou plusieurs interfaces. Pour les décrire simplement, les interfaces sont des définitions de la façon dont une classe doit répondre. Une interface définit les méthodes, les propriétés et les événements qu'une classe doit implémenter ainsi que le type de paramètres que chaque membre doit recevoir et retourner, mais elle laisse l'implémentation spécifique de ces membres à la charge de la classe d'implémentation.

Les interfaces multiples permettent l'évolution des systèmes de composants logiciels sans endommager le code existant.

Pour plus d'informations, consultez Polymorphisme d'interface ou Comment : créer et implémenter des interfaces.

Méthodes : paramètres ByVal et ByRef

Dans Visual Basic 6.0, les arguments sont passés à une méthode par valeur ou par référence. Sauf spécification explicite des paramètres de méthode par le mot clé **ByVal** ou **ByRef**, les arguments sont implicitement passés par référence (**ByRef**).

Dans Visual Basic 2005, les arguments sont passés à une méthode par valeur ou par référence. Sauf spécification explicite des paramètres de méthode par le mot clé **ByVal** ou **ByRef**, les arguments sont implicitement passés par valeur (**ByVal**).

Pour plus d'informations, consultez Passage d'arguments par valeur et par référence.

Méthodes : Surcharge

Dans Visual Basic 6.0, la surcharge de méthode n'est pas prise en charge.

Dans Visual Basic 2005, la surcharge **Method** est utilisée pour implémenter le polymorphisme ad hoc dans un programme Visual Basic. Une méthode est surchargée lorsque plusieurs versions de la méthode sont définies dans une classe. Les versions surchargées diffèrent au niveau de leurs paramètres et types de retours.

Pour plus d'informations, consultez Propriétés et méthodes surchargées et Polymorphisme dans Visual Basic.

Méthodes : Substitution

Dans Visual Basic 6.0, les méthodes ne peuvent pas être substituées.

Dans Visual Basic 2005, vous pouvez utiliser le mot clé **Overrides** afin de substituer une méthode dans une classe dérivée et fournir une implémentation différente d'une méthode de la classe de base de la classe dérivée.

Pour plus d'informations, consultez Substitution de propriétés et de méthodes.

Méthodes : Retour de résultats

Dans Visual Basic 6.0, un nom de méthode **Function** est utilisé comme nom d'une variable pour retourner le résultat de la méthode **Function**.

Dans Visual Basic 2005, vous utilisez le mot clé **Return** pour retourner un résultat de méthode **Function**.

Utilisez le **Return** pour retourner le contrôle au code qui a appelé une méthode **Sub**, **Function** ou **Property**.

Pour plus d'informations, consultez Return, instruction (Visual Basic).

My

Dans Visual Basic 2005, la nouvelle fonctionnalité **My** active la programmation orientée objet rapide avec Visual Basic en fournissant des points d'entrée aux classes et aux fonctions .NET Framework fréquemment utilisées. **My** fournit de nouvelles classes .NET Framework de haut niveau qui rassemblent des fonctionnalités connexes dans des API basées sur tâche.

Pour plus d'informations, consultez Développement avec My.

Mots clés MyBase, MyClass et Me

Dans Visual Basic 6.0, le mot clé **Me** se comporte comme une variable déclarée implicitement. Lorsqu'une classe possède plusieurs instances, **Me** offre un moyen de faire référence à l'instance spécifique de la classe où le code s'exécute.

Dans Visual Basic 2005, le mot clé **MyBase** offre un moyen de faire référence à la classe de base de l'instance de classe actuelle.

Le mot clé **MyClass** offre un moyen de faire référence à l'instance de classe actuelle sans substitutions.

Le mot clé **Me** permet de faire référence à l'instance spécifique d'une classe ou d'une structure dans laquelle le code est en train de s'exécuter.

Pour plus d'informations, consultez Me, My, MyBase et MyClass dans Visual Basic.

Mot clé New

Dans Visual Basic 6.0, l'utilisation inappropriée du mot clé **New** peut provoquer des comportements de réinitialisation inhabituels susceptibles d'induire des erreurs et une utilisation excessive de la mémoire.

Dans Visual Basic 2005, le mot clé **New** se contente d'allouer de l'espace pour l'objet. Il ne provoque aucun comportement de réinitialisation inhabituel.

Utilisez une clause **New** dans une instruction de déclaration ou une instruction d'assignation. Lorsque l'instruction est exécutée, elle appelle une méthode de constructeur de la classe spécifiée, en passant tous les arguments fournis dans la clause **New**. Cela signifie que vous pouvez appliquer des contraintes de données à partir du moment où l'objet est instancié.

Une classe possède un ou plusieurs constructeurs. Les constructeurs sont les méthodes dans une méthode nommée **New**. Plusieurs constructeurs sont définis par surcharge de la méthode **New** dans la classe. Les méthodes de constructeur surchargées sont décrites comme des constructeurs paramétrés.

Pour plus d'informations, consultez New (Visual Basic).

Durée de vie de l'objet

Dans Visual Basic 6.0, la durée de vie des objets est déterministe ; chaque instance d'objet gère un décompte de références. Lorsque la dernière référence à une instance est diffusée et que le compte atteint la valeur zéro, l'objet est immédiatement terminé.

Dans Visual Basic 2005, la durée de vie des objets n'est pas déterministe ; un destructeur n'est pas nécessairement appelé dès la diffusion de la dernière référence. En effet, le Common Language Runtime gère une arborescence des références au lieu de décomptes de références individuelles.

Le garbage collector trace cette arborescence des références à l'arrière-plan. S'il trouve un objet ou un groupe d'objets qui n'ont aucune référence d'aucun code en cours d'exécution, le garbage collector appelle les destructeurs de tous ces types d'objets.

Il est impossible de prévoir ni l'ordre des destructions, ni le temps qu'il faut au garbage collector pour tracer l'arborescence des références.

Pour plus d'informations, consultez Durée de vie dans Visual Basic ou
Durée de vie d'un objet : création et destruction des objets.

Surcharge d'opérateur

Dans Visual Basic 6.0, la surcharge d'opérateur n'est pas prise en charge.

Dans Visual Basic 2005, la surcharge d'opérateur permet au comportement d'un opérateur standard (tel que ***** **<>** ou **And**) d'être défini pour une classe ou une structure définie par l'utilisateur.

Pour plus d'informations, consultez Comment : définir un opérateur de conversion.

Option Strict

Dans Visual Basic 6.0, la vérification de type permissive est utilisée au moment de l'exécution.

Dans Visual Basic 2005, vous pouvez utiliser l'instruction **OPTION STRICT ON** pour activer la vérification de type stricte au moment du design plutôt que la vérification de type permissive au moment de l'exécution. Par défaut, **OPTION STRICT OFF** est utilisé.

Pour les conversions de types de données, le compilateur Visual Basic peut fonctionner avec une sémantique de type stricte ou permissive. Si la sémantique de type stricte est active (**OPTION STRICT ON**), seules les conversions étendues sont implicitement autorisées et les conversions restrictives doivent être explicites. Si la sémantique de type permissive (**OPTION STRICT OFF**) est active, vous pouvez tenter d'effectuer toutes les conversions étendues et restrictives de manière implicite. Ces sémantiques s'appliquent aux conversions entre tous les types de données, y compris les types d'objets.

Pour plus d'informations, consultez Vérification du type dans Visual Basic.

Syntaxe des propriétés

Dans Visual Basic 6.0, utilisez les instructions **Property Get**, **Property Let** et **Property Set** pour obtenir et définir des valeurs de propriété.

Dans Visual Basic 2005, une syntaxe de déclaration de propriété unifiée qui englobe les procédures destinées à obtenir et à définir la valeur de la propriété est utilisée. Cette syntaxe garantit la cohérence d'attributs de propriété tels que le niveau d'accès et la surcharge.

Pour plus d'informations, consultez Modifications des propriétés pour les utilisateurs de Visual Basic 6.0.

Interfaces publiques

Dans Visual Basic 6.0, l'interface publique d'une classe est constituée de l'ensemble de ses membres publics.

Dans Visual Basic 2005, l'interface publique d'une classe est constituée de l'ensemble de ses membres publics, mais une classe abstraite peut être utilisée pour implémenter le polymorphisme de sous-type dans un programme Visual Basic.

Une classe abstraite est semblable au **Interface** du .NET Framework. Une classe abstraite est une classe conçue uniquement pour servir de classe de base à des classes dérivées. Les classes abstraites sont souvent utilisées pour représenter des concepts ou des entités abstraites.

Une classe abstraite ne peut pas être instanciée elle-même ; elle doit être héritée. Certains, voire tous les membres de la classe peuvent ne pas être implémentés, et c'est à la classe qui hérite de fournir cette implémentation. Les membres implémentés peuvent toujours être substitués, et la classe qui hérite peut toujours implémenter des interfaces supplémentaires ou d'autres fonctionnalités.

Pour plus d'informations, consultez MustInherit.

Propriétés en lecture seule

Dans Visual Basic 6.0, l'exclusion de la méthode **Set** dans une procédure **Property** crée une propriété en lecture seule.

Dans Visual Basic 2005, vous devez à la fois appliquer le mot clé **ReadOnly** à une propriété et exclure la clause **Set** de la propriété pour en faire une propriété en lecture seule.

Vous pouvez également appliquer le mot clé **ReadOnly** à un champ pour en faire un champ en lecture seule.

Pour plus d'informations, consultez ReadOnly (Visual Basic).

Réflexion

Dans Visual Basic 6.0, la réflexion n'est pas prise en charge.

Dans Visual Basic 2005, les classes de l'espace de noms System.Reflection de la bibliothèque de classes du .NET Framework peuvent être utilisées pour obtenir des informations à propos de types tels que les classes, les interfaces et les types valeur au moment de l'exécution et créer des instances de type pour les appeler et y accéder.

Pour plus d'informations, consultez Espaces de noms de réflexion dans Visual Studio.

Membres de classe et de structure partagées

Dans Visual Basic 2005, le nouveau mot clé **Shared** peut être appliqué aux instructions **Property**, **Sub**, **Dim**, **Function**, **Operator**et **Event** pour les partager avec toutes les instances d'une classe ou structure. Les membres partagés d'une classe sont accessibles sans instanciation de la classe.

Vous pouvez accéder aux membres partagés par le nom d'une classe plutôt que via une instance de la classe.

Pour plus d'informations, consultez Membres partagés en Visual Basic.

Fichiers sources

Dans Visual Basic 2005, les programmes se composent d'un ou de plusieurs fichiers sources. Les fichiers sources sont stockés dans un type de fichier spécial qui possède une extension de fichier .vb. Les instructions de type pour des types tels que les classes, les structures, les interfaces, les énumérations et les délégués, ainsi que les instructions pour les modules standard, sont stockées dans les fichiers de code source de programme. Visual Basic ne limite pas un fichier source à définir uniquement un seul type et ne nécessite pas que le nom du fichier source corresponde à un type défini dans le fichier source.

Les instructions de type partiel autorisent une classe, ou une structure, à être définie à travers plusieurs fichiers sources. Lorsqu'un programme à plusieurs fichiers sources est compilé, tous les fichiers sources sont traités ensemble, comme si tous les fichiers sources étaient concaténés dans un même fichier avant d'être traités.

Pour plus d'informations, consultez Structure d'un programme Visual Basic ou Partial (Visual Basic).

Types définis par l'utilisateur

Dans Visual Basic 6.0, l'instruction **Type** est utilisée pour combiner des variables de plusieurs types différents en un seul type défini par l'utilisateur, ou UDT.

Dans Visual Basic 2005, les instructions **Structure** et **Enum** sont utilisées pour définir des types valeur définis par l'utilisateur.

Utilisez les instructions **Class**, **Interface** et **Delegate** pour définir des types référence définis par l'utilisateur.

Pour plus d'informations, consultez Structures constituées à partir de vos propres types de données.

Déclaration des variables

Dans Visual Basic 6.0, vous pouvez déclarer des variables de types différents dans la même instruction. Si vous ne spécifiez pas le type de données d'une variable dans l'instruction, son type sera **Variant** par défaut.

Dans Visual Basic 2005, vous pouvez déclarer plusieurs variables du même type de données sans devoir spécifier le type de données de chaque variable dans l'instruction.

Pour plus d'informations, consultez Déclaration de variable dans Visual Basic.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau en Visual Basic 2005, les constructions de code orientées objet sont converties en leur équivalent Visual Basic 2005 le plus proche, le plus rapidement possible. Le code qui résulte ne doit pas être pris comme exemple de la manière avec laquelle effectuer de la programmation orientée objet dans Visual Basic 2005.

Si vous avez l'intention d'étendre une application mise à niveau de sorte qu'elle utilise des constructions orientées objet Visual Basic 2005, il peut être judicieux de modifier d'abord le code mis à niveau. Les pages suivantes devraient vous aider à entamer ce travail.

Classes pour les utilisateurs de Visual Basic 6.0

Les classes sont les blocs de construction d'une application orientée objet.

Comment : convertir une collection personnalisée en collection typée Visual Basic

Visual Basic 2005 fournit plusieurs options pour créer des classes typées à partir des collections personnalisées de Visual Basic 6.0.

Comment : convertir un type défini par l'utilisateur en structure Visual Basic

L'instruction **Type** de Visual Basic 6.0 est mise à niveau avec l'instruction **Structure** de Visual Basic 2005.

Comment : convertir du code Implements en nouvelle forme d'héritage

Des versions plus récentes de Visual Basic offrent deux types d'héritages, à l'aide de l'instruction **Implements** et de l'instruction **Inherits**.

Voir aussi

Tâches

Programmation orientée objet, exemple

Concepts

Classes pour les utilisateurs de Visual Basic 6.0
Modifications orientées objet pour les modules en Visual Basic
Modifications de l'interface pour les utilisateurs de Visual Basic 6.0
Héritage pour les utilisateurs de Visual Basic 6.0

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0
Aide pour les utilisateurs de Visual Basic 6.0

Classes pour les utilisateurs de Visual Basic 6.0

Les classes sont les blocs de construction d'une application orientée objet. Une classe est une construction de programmation qui représente un objet dans le système. La conception d'une application requiert la conception des objets impliqués, des informations contenues par les objets et des opérations que les objets peuvent exécuter. Pour plus d'informations sur les classes et la programmation orientée objet, consultez Programmation orientée objet dans Visual Basic.

Modules de classe

Visual Basic 6.0

Visual Basic 6.0 fournit la prise en charge de la définition des classes via le module de classe. La définition d'une classe est stockée dans un type de fichier spécial, le fichier .cls. Une classe est définie dans chaque module de classe. Une partie de la définition de la classe est incorporée dans le fichier .cls et peut uniquement être modifiée via le système de projet.

Visual Basic 2005

Dans Visual Basic 2005, une classe est définie par l'instruction Class (Visual Basic), et non par un nom de fichier. Les classes dans Visual Basic 2005 sont définies dans les fichiers de code source, c'est-à-dire les fichiers .vb. Plusieurs classes peuvent être définies dans un fichier. La définition de classe tout entière est visible sous forme de texte brut dans le code source.

Classes globales

Visual Basic 6.0

Lorsque vous créez une nouvelle classe dans Visual Basic 6.0, **GlobalSingleUse** et **GlobalMultiUse** figurent parmi les valeurs possibles de la propriété **Instancing**. Ceci indique que d'autres composants peuvent appeler les propriétés et méthodes de la nouvelle classe comme s'il s'agissait de membres partagés. Une instance de la classe est créée implicitement lors du premier appel de l'un de ses membres.

Visual Basic 2005

Visual Basic 2005 ne prend pas en charge la propriété **Instancing**. Visual Basic 2005 fournit les mêmes fonctionnalités en vous permettant d'exposer des membres de module standard. Vous pouvez accéder aux propriétés et méthodes partagées d'une classe à l'aide de l'instruction **Imports**. Il est possible d'obtenir le même effet que **GlobalMultiUse** par l'accès à une classe **Public** et au constructeur approprié.

Données, classes

Visual Basic 6.0

Visual Basic 6.0 prend également en charge les classes Source de données et Data Consumer Complex.

Visual Basic 2005

Il n'y a aucun équivalent direct dans Visual Basic 2005. Pour plus d'informations sur les classes qui manipulent des données, consultez Comment : établir une connexion à des données dans un objet.

Membres par défaut

Visual Basic 6.0

Dans Visual Basic 6.0, vous pouvez spécifier une donnée membre ou une méthode particulière comme le membre par défaut d'une classe.

Visual Basic 2005

Dans Visual Basic 2005, un membre par défaut d'une classe ou d'une structure peut être uniquement une propriété qui accepte un ou plusieurs arguments. Cette mesure a pour but d'assurer l'interopérabilité avec d'autres langages de programmation. Pour plus d'informations, consultez Comment : déclarer et appeler une propriété par défaut en Visual Basic.

Durée de vie de l'objet

Visual Basic 6.0

La durée de vie d'un objet est déterminée par la création et la fin de l'instance de l'objet. Le programme détermine le moment de création des objets qu'il déclare, mais la fin implique un mécanisme plus complexe.

Dans Visual Basic 6.0, chaque instance d'objet gère un décompte de références. Lorsque la dernière référence d'une instance est libérée et que le compteur revient à zéro, l'objet est immédiatement terminé.

Visual Basic 2005

Dans Visual Basic 2005, un destructeur n'est pas nécessairement appelé dès que la dernière référence est libérée. En effet, le Common Language Runtime gère une arborescence des références au lieu de décomptes de références individuelles. Le garbage collector trace cette arborescence des références à l'arrière-plan. S'il détecte un objet ou un groupe d'objets qui ne possèdent aucune référence dans le code actuellement en cours d'exécution, il appelle les destructeurs de ces objets. Il est impossible de prévoir ni l'ordre des destructions, ni le temps qu'il faut au garbage collector pour tracer l'arborescence des références. C'est pour cette raison que la durée de vie d'un objet est indéterminée. Pour plus d'informations, consultez Durée de vie d'un objet : création et destruction des objets et Garbage collection.

Mettre à niveau des options

L'Assistant Mise à niveau crée une mise à niveau simple de modules de classe avec les modifications minimales apportées au code. En plus des modifications répertoriées ci-dessus, les programmeurs Visual Basic possèdent plusieurs nouvelles constructions parmi lesquelles choisir qui sont est particulièrement appropriées pour la programmation orientée objet. Quelques-unes sont répertoriées ci-dessous.

- **Les constructeurs** Les constructeurs remplacent l'utilisation de **Class_Initialize** de Visual Basic 6.0. Pour plus d'informations, consultez Modifications de Class_Initialize pour les utilisateurs de Visual Basic 6.0.
- **Les propriétés indexées par défaut** Les propriétés indexées par défaut sont largement utilisées dans le .NET Framework pour accéder aux éléments de collection. Pour plus d'informations, consultez Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0.
- **La lecture seule** Dans Visual Basic 6.0, une propriété est implicitement en lecture seule s'il n'y a aucune procédure **Get**. Visual Basic 2005 requiert que le mot clé **ReadOnly** pour déclarer explicitement ce comportement. Pour plus d'informations, consultez Property, instruction.
- **La liaison tardive** Visual Basic 6.0 fournissant la liaison tardive par défaut, le moyen le plus sûr de mettre à niveau le code sans provoquer d'erreurs est d'affecter **Off** à **Option Strict**. La modification de **Option Strict** en **On** applique la sécurité de type au moment de la compilation et peut exposer des erreurs subtiles au moment de la compilation plutôt qu'au moment de l'exécution. Pour plus d'informations, consultez Option Strict, instruction.
- **La surcharge des méthodes** la surcharge des méthodes réduit le nombre de membres nommés d'une classe. Cela rend la classe plus lisible et plus facile à programmer. Pour plus d'informations, consultez Modifications des procédures pour les utilisateurs de Visual Basic 6.0.
- **Le nom de fichier** Visual Basic 6.0 requiert que le nom de classe et le nom de fichier correspondent. Dans Visual Basic 2005, le programmeur contrôle les deux et peut définir plusieurs classes dans un fichier source. Le nombre de classes contenu dans un fichier est généralement traité par une norme de codage.
- **La valeur ByRef** La valeur par défaut des paramètres dans Visual Basic 6.0 est **ByRef**, par conséquent, la mise à niveau la plus sûre est **ByRef**. Pour plus d'informations, consultez ByVal.
- **Le mot clé Return** Dans Visual Basic 6.0, le nom de fonction est utilisé comme variable pour retourner la valeur de la fonction. Visual Basic 2005 fournit le mot clé **Return** explicite pour retourner des valeurs fonctionnelles. Pour plus d'informations, consultez Return, instruction (Visual Basic).
- **La surcharge d'opérateur** Vous pouvez désormais définir la signification de l'addition (**+**), de la soustraction (**-**) et d'autres opérateurs pour les classes que vous définissez. Pour plus d'informations, consultez Operator, instruction.
- **Les nouveaux opérateurs** L'opérateur **+=** peut raccourcir et simplifier une ligne de code. Pour plus d'informations, consultez Opérateurs arithmétiques (Visual Basic).
- **La définition en ligne de nouvelles valeurs** Dans Visual Basic 2005, la syntaxe permet de déclarer et de définir la valeur d'une nouvelle variable en une seule ligne de code. Cela signifie que vous pouvez appliquer des contraintes de données à partir du moment où l'objet est instancié. Pour plus d'informations, consultez Dim, instruction (Visual Basic).

La réécriture du code n'est pas sûre. Lorsque le code est modifié, il doit être testé une nouvelle fois. De même, certains mots clés dans Visual Basic 2005 n'agissent pas exactement de la même façon que dans Visual Basic 6.0. Pour plus d'informations, consultez Mise à niveau d'applications créées dans des versions précédentes de Visual Basic.

Voir aussi

Référence

Structure, instruction
Property, instruction
Option Strict, instruction
Return, instruction (Visual Basic)

Impossible de combiner 'ByVal' et 'ByRef'
ByRef

Concepts

Liaison anticipée et liaison tardive
Modifications de Class_Initialize pour les utilisateurs de Visual Basic 6.0

Autres ressources

Création et utilisation d'objets
Présentation des classes

Modifications de Class_Initialize pour les utilisateurs de Visual Basic 6.0

Un constructeur est une méthode qui est exécutée lorsqu'un objet est créé. Les constructeurs sont nouveaux dans Visual Basic. Ils combinent le comportement de l'instruction **New** et de la méthode **Class_Initialize** de Visual Basic 6.0 , toutefois, ils permettent une création plus flexible et mieux contrôlée des instances de classe. Pour plus d'informations sur la création d'objets, consultez Création et utilisation d'objets.

Class_Initialize

Visual Basic 6.0

Visual Basic 6.0 permet une prise en charge du concept de constructeur via la méthode **Class_Initialize**. Cette méthode est privée et n'autorise pas de paramètres. Cette méthode est automatiquement appelée lorsqu'une nouvelle instance de la classe est créée. Un appel au mot clé **New** appelle la méthode **Class_Initialize**, s'il existe.

Supposez que vous souhaitez spécifier la croissance annuelle d'une arborescence avec une valeur par défaut de cinq années. Le code de la classe se présenterait de la manière suivante :

```
' Visual Basic 6.0
Private mvarYearlyGrowth As Integer

Public Property Get YearlyGrowth() As Integer
    YearlyGrowth = mvarYearlyGrowth
End Property

Public Property Let YearlyGrowth(ByVal newValue As Integer)
    mvarYearlyGrowth = newValue
End Property

Private Sub Class_Initialize()
    mvarHeight = 5
End Sub
```

Le code permettant de créer une arborescence et de définir la propriété se présenterait de la manière suivante :

```
Dim growingTree As New Tree
growingTree.YearlyGrowth = 10
```

Visual Basic 2005

Visual Basic 2005 prend en charge des constructeurs via le mot clé **New**. Dans ce cas, le code supplémentaire permettant de définir la propriété `YearlyGrowth` n'est pas requis. Un appel au mot clé **New** appelle l'une des méthodes **New** surchargées dans la classe. Si la classe ne contient aucune méthode **New**, une classe sans paramètres est créée par le compilateur.

VB

```
Option Strict On
```

VB

```
Public Class Tree
  Private yearlyGrowthValue As Integer = 5

  Public Property YearlyGrowth() As Integer
      Get
          Return yearlyGrowthValue
      End Get
      Set(ByVal Value As Integer)
          yearlyGrowthValue = Value
      End Set
  End Property
```

```
  Public Sub New(ByVal newYearlyGrowth As Integer)
      Me.YearlyGrowth = newYearlyGrowth
  End Sub
End Class
```

Le code permettant de créer une arborescence et de définir la propriété se présenterait de la manière suivante :

VB

```
Dim growingTree As New Tree(10)
```

Dans cet exemple, il n'est pas possible de créer un `Tree` sans spécifier la croissance annuelle. Pour cela, vous pouvez ajouter une autre méthode `New` à la classe qui n'a pas de paramètres. Cette opération est appelée *surcharge* du constructeur.

Nouveau mot clé

Visual Basic 6.0

Dans Visual Basic 6.0, cette ligne de code n'est généralement pas recommandée :

```
Dim growingTree As New Tree
```

Cela est dû au fait que chaque fois que la variable `growingTree` est accédée, le compilateur vérifie si la valeur est **Nothing**. Le cas échéant, une nouvelle instance est créée et assignée à `growingTree`. Cette opération est non seulement inefficace, mais provoque des erreurs de programmation.

Visual Basic 2005

Dans Visual Basic 2005, l'instance n'est pas activée et aucune nouvelle instance n'est créée. La seule façon de créer une nouvelle instance est d'exécuter une ligne de code avec les mots clé **New** ou **As New**. En fait, la ligne de code ci-dessus est désormais recommandée pour gérer la création d'une nouvelle instance.

Suggestions de mise à niveau

L'Assistant Mise à niveau met à niveau la méthode **Class_Initialize** vers :

VB

```
'UPGRADE_NOTE: Class_Initialize was upgraded to Class_Initialize_Renamed. Click for more: m
s-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1061"'

Private mvarHeight As Short
Private Sub Class_Initialize_Renamed()
    mvarHeight = 0
End Sub

Public Sub New()
    MyBase.New()
    Class_Initialize_Renamed()
End Sub
```

Vous pouvez peut-être éliminer l'appel à `Class_Initialize_Renamed` et déplacer le code directement dans le constructeur. Cela peut rendre votre code plus lisible étant donné que cela élimine un appel inutile à une procédure. Si chaque appel à **New** est couramment suivi de la définition d'un groupe particulier de propriétés, envisagez d'ajouter un constructeur qui définisse ces propriétés. Le code Visual Basic 6.0 peut être remplacé par ce code dans Visual Basic 2005 :

VB

```
Public Sub New(ByVal newYearlyGrowth As Integer)
    Me.YearlyGrowth = newYearlyGrowth
End Sub
```

Voir aussi

Tâches

Comment : utiliser le nouveau mot clé

Autres ressources

Création et utilisation d'objets

Modifications des propriétés de Visual Basic

Les champs et les propriétés fonctionnent ensemble pour fournir l'accès aux données d'une classe. Les champs sont les variables situées dans une définition de classe et contiennent les données de l'objet. Les propriétés fournissent les méthodes **Get** et **Set** pour autoriser l'accès aux données de l'instance de classe. Pour plus d'informations, consultez Propriétés et procédures Property.

Syntaxe des propriétés

Visual Basic 6.0

Le contrôle **Property** est une combinaison d'un objet **Get** et d'un objet **Set**. Dans Visual Basic 6.0, ces deux méthodes sont séparées :

```
' Visual Basic 6.0
Private mvarYearlyGrowth As Integer
Public Property Get YearlyGrowth() As Integer
    YearlyGrowth = mvarYearlyGrowth
End Property

Public Property Let YearlyGrowth(ByVal newValue As Integer)
    mvarYearlyGrowth = newValue
End Property
```

Visual Basic 2005

Les modifications de propriété de Visual Basic 6.0 vers Visual Basic 2005 sont essentiellement syntaxiques. La syntaxe Visual Basic 2005 déplace les méthodes **Get** et **Set** dans un bloc de code pour la propriété, au lieu de deux méthodes indépendantes. Visual Basic 2005 ne fait aucune distinction entre les propriétés qui sont des types référence et les propriétés qui sont des types valeur, par conséquent, il n'y a qu'une méthode **Set** et il n'y a plus de méthode **Let**. Le champ et la propriété de `YearlyGrowth` auraient l'apparence suivante dans Visual Basic 2005 :

VB

```
Private yearlyGrowthValue As Integer = 5

Public Property YearlyGrowth() As Integer
    Get
        Return yearlyGrowthValue
    End Get
    Set(ByVal Value As Integer)
        yearlyGrowthValue = Value
    End Set
End Property
```

Dans Visual Basic 6.0 et dans Visual Basic 2005, la classe contient un champ, `mvarYearlyGrowth` ou `yearlyGrowthValue`, qui contient les données.

Suggestions de mise à niveau

L'**Assistant Mise à niveau** met à niveau une propriété avec :

VB

```
Private mvarYearlyGrowth As Short

Public Property YearlyGrowth() As Short
    Get
        YearlyGrowth = mvarYearlyGrowth
    End Get
    Set(ByVal Value As Short)
        mvarYearlyGrowth = Value
    End Set
End Property
```

L'**Integer** Visual Basic 6.0 est mis à niveau avec **Short** dans Visual Basic 2005. Cela pourrait provoquer des problèmes d'incompatibilité de type lorsqu'un autre code essaie d'assigner un **Integer** dans Visual Basic 2005 à la propriété. Vous devrez peut-être examiner les types convertis pour les champs et les propriétés afin d'éviter d'autres conflits de sécurité de type.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Property, instruction

Autres ressources

Modifications des propriétés pour les utilisateurs de Visual Basic 6.0

Modifications des méthodes pour les utilisateurs de Visual Basic 6.0

Une méthode est une déclaration de procédure **Sub** ou **Function**. Les méthodes font accomplir des tâches à une classe. Avec les propriétés, elles exposent le comportement et les données d'une classe.

Déclarations de méthodes

Visual Basic 6.0

Dans Visual Basic 6.0, les définitions de classe peuvent contenir des méthodes **Sub** et **Function**. Ces méthodes peuvent être **Public**, **Private** ou **Friend**. En voici deux exemples :

```
Public Sub Draw(theForm As Form)
    ' Add code here to draw a tree on the form.
End Sub

Public Function GrowOneYear() As Integer
    mvarHeight = mvarHeight + 10
    GrowOneYear = mvarHeight
End Function
```

Visual Basic 2005

Visual Basic 2005 prend également en charge des méthodes **Sub** et **Function** qui sont **Public**, **Private** et **Friend**. **ByRef** ou **ByVal** est requis dans tous les paramètres de méthode. S'il n'est pas explicitement ajouté, **ByVal** est ajouté par l'**éditeur de code**.

Le code mis à niveau de Visual Basic 6.0 sera mis à niveau vers **ByRef**, comme tel était le cas par défaut dans Visual Basic 6.0 :

VB

```
Public Sub Draw(ByRef theForm As System.Windows.Forms.Form)
    ' Add code here to draw a tree on the form.
End Sub
```

Surcharge des méthodes

Visual Basic 6.0

Dans Visual Basic 6.0, une classe ne peut contenir qu'une méthode avec un nom donné.

Visual Basic 2005

Dans Visual Basic 2005, les méthodes peuvent être surchargées. Une méthode est surchargée lorsque plusieurs versions de la méthode sont définies dans la classe. Les versions surchargées diffèrent au niveau de leurs paramètres et types de retours. Cela peut simplifier l'interface d'un objet en le rendant plus petit. L'exemple suivant illustre deux surcharges de la méthode `Draw`. Pour plus d'informations, consultez Propriétés et méthodes surchargées.

VB

```
Public Sub Draw(ByRef theForm As System.Windows.Forms.Form)
    ' Add code here to draw a tree on the form.
End Sub
```

VB

```
Public Sub Draw(ByVal surface As System.Drawing.Graphics)
    ' Add code here to draw a tree on the graphics surface.
End Sub
```

Mot clé Return

Visual Basic 6.0

Dans Visual Basic 6.0, le nom de fonction est utilisé en tant que variable pour retourner la valeur de la fonction.

```
Public Function GrowOneYear() As Integer
    heightValue = heightValue + 10
    GrowOneYear = heightValue
End Function
```

Visual Basic 2005

Visual Basic 2005 inclut l'instruction explicite **Return** pour retourner des valeurs fonctions. La fonction `GrowOneYear` dans Visual Basic 2005 a l'aspect suivant :

VB

```
Public Function GrowOneYear() As Integer
    heightValue += 10
    Return heightValue
End Function
```

Pour plus d'informations, consultez Return, instruction (Visual Basic).

Suggestions de mise à niveau

L'Assistant Mise à niveau produit ce code à partir de la méthode `Draw` :

VB

```
Public Sub Draw(ByRef theForm As System.Windows.Forms.Form)
    ' Add code here to draw a tree on the form.
End Sub
```

Certaines techniques à envisager dans les méthodes mises à niveau incluent :

- Envisagez l'utilisation des méthodes surchargées pour les méthodes portant des noms et présentant un comportement similaire, mais qui varient au niveau des types de retours et de paramètres.
- Vérifiez la conversion de la portée (**Public**, **Private**, **Friend**) et du type de retour (**Short**, **Integer**, etc.) des méthodes.
- Envisagez d'utiliser l'instruction **Return** au lieu du nom de fonction pour retourner la valeur de la fonction.

Voir aussi

Référence

Sub, instruction (Visual Basic)
Function, instruction (Visual Basic)
Return, instruction (Visual Basic)

Héritage pour les utilisateurs de Visual Basic 6.0

L'héritage vous permet de créer de nouvelles classes à partir de classes existantes. L'héritage peut simplifier le design de l'application en fournissant une structure de relation entre les différentes classes. Il prend également en charge la réutilisation de code, le comportement des classes nouvelles ou différentes devant être codé.

Héritage de l'interface

Visual Basic 6.0

Visual Basic 6.0 prend en charge l'héritage de l'interface, traité avec plus de détail dans Modifications de l'interface pour les utilisateurs de Visual Basic 6.0.

Visual Basic 2005

Visual Basic 2005 prend en charge l'héritage d'interfaces avec l'Interface, instruction (Visual Basic) et l'Implements, instruction. Pour plus d'informations sur les interfaces, consultez Interfaces en Visual Basic.

Héritage d'implémentation

Lorsqu'une nouvelle classe est créée via l'héritage d'implémentation, la classe nouvelle bénéficie automatiquement de tous les membres et de l'implémentation de la classe de base. La classe existante est appelée "classe de base". La nouvelle classe est la "classe dérivée". Des membres supplémentaires peuvent être ajoutés à la classe. Le comportement de la classe de base peut être modifié en écrivant du code dans la nouvelle classe. Cette technique est appelée substitution.

Vous pouvez créer des classes en héritant des classes que vous créez, des classes figurant dans les références que vous ajoutez à votre projet ou des objets dans le .NET Framework. De nombreuses classes du .NET Framework sont liées par héritage. Par exemple, la classe TextBox hérite de la classe System.Windows.Forms.TextBoxBase.

Visual Basic 6.0

Visual Basic 6.0 ne prend pas en charge l'héritage d'implémentation.

Visual Basic 2005

L'héritage est déclaré dans Visual Basic 2005 par l'instruction **Inherits**. Dans cet exemple, la classe dérivée `Oak` hérite d'une classe de base `Tree`.

VB

```
Public Class Oak
    Inherits Tree
    ' Add code here to extend or
    ' modify the behavior of the Tree class.
End Class
```

MustInherit et NotInheritable

Visual Basic 6.0

Dans l'héritage de l'interface dans Visual Basic 6.0, toute classe peut agir comme une classe de base avec interface. Il n'existe aucune syntaxe permettant d'empêcher toute classe d'agir comme une classe de base avec interface. De même, il n'existe aucune syntaxe permettant d'exiger qu'une classe agisse uniquement comme une classe de base avec interface.

Visual Basic 2005

Avec l'héritage d'implémentation, Visual Basic 2005 définit deux modificateurs de classe, MustInherit et NotInheritable. Ces modificateurs laissent le programmeur contrôler les relations d'héritage dans une application.

Le modificateur **MustInherit** sur une déclaration de classe indique une classe qui ne peut pas être instanciée :

VB

```
Public MustInherit Class BaseClass
End Class
```

Cela signifie que `BaseClass` ne peut pas apparaître après le mot clé **New**. Seules classes qui héritent de `BaseClass` et n'ont pas

non plus de modificateur **MustInherit** peuvent être instanciées. Dans la littérature orientée objet et dans d'autres langages orientés objet, une classe **MustInherit** est appelée classe *abstraite*. Une classe qui n'est pas une classe **MustInherit**, c'est-à-dire une classe qui peut être instanciée, est appelée classe *concrète*.

Un concept lié est la classe sealed, c'est-à-dire une classe qui ne peut pas servir de classe de base. Le mot clé **NotInheritable** sur une définition de classe indique l'état suivant :

VB

```
Public NotInheritable Class DerivedClass
End Class
```

Une classe **NotInheritable** marque le bord d'une hiérarchie d'héritage.

Pour plus d'informations, consultez Héritage en Visual Basic.

Suggestions de mise à niveau

Les classes qui agissent comme des interfaces en Visual Basic 6.0 sont mises à niveau vers des interfaces dans Visual Basic 2005. Considérez l'exemple de classe de base et de classe dérivée à partir de Visual Basic 6.0 suivant :

```
' Contents of class BaseClass
Public Sub BaseMethod()
End Sub

' Contents of class DerivedClass
Implements BaseClass
Private Sub BaseClass_BaseMethod()
End Sub
```

L'Assistant Mise à niveau produit le code mis à niveau suivant :

VB

```
Option Strict Off
Option Explicit On
Interface _BaseClass
    Sub BaseMethod()
End Interface

Friend Class BaseClass
    Implements _BaseClass
    Public Sub BaseMethod() Implements _BaseClass.BaseMethod
    End Sub
End Class

Friend Class DerivedClass
    Implements _BaseClass
    Private Sub BaseClass_BaseMethod() Implements _BaseClass.BaseMethod
    End Sub
End Class
```

À l'aide de l'héritage plutôt que des interfaces, le code mis à niveau peut être modifié comme suit :

VB

```
Friend Class BaseClass
    Public Sub BaseMethod()
        ' Add code here to define BaseMethod.
    End Sub
End Class

Friend Class DerivedClass
    Inherits BaseClass
End Class
```

Cela élimine un niveau d'indirection, `_BaseClass`. De plus, cela élimine une définition de méthode, `BaseClass_BaseMethod`, la méthode étant héritée de la classe de base et ne devant pas être à nouveau codée. Si le programmeur souhaitait un comportement différent pour la classe dérivée, `BaseMethod` peut être substituée comme indiqué :

VB

```
Friend Class BaseClass
    Public Overridable Sub BaseMethod()
        ' Add code here to define BaseMethod.
    End Sub
End Class

Friend Class DerivedClass
    Inherits BaseClass
    Public Overrides Sub BaseMethod()
        ' Add code here to define behavior for DerivedClass.
    End Sub
End Class
```

Certaines des techniques à considérer dans les interfaces mises à niveau incluent :

- Le remplacement des instructions **Interface** par des instructions **Inherits**.
- L'implémentation de la classe de base comme une interface plutôt que comme une classe qui implémente une interface. Pour plus d'informations, consultez Modifications de l'interface pour les utilisateurs de Visual Basic 6.0.
- L'implémentation des classes de base comme des classes **MustInherit**.
- L'implémentation des classes dérivées comme **NotInheritable**.

Voir aussi

Autres ressources

Héritage en Visual Basic
Interfaces en Visual Basic

Modifications des événements et des exceptions pour les utilisateurs de Visual Basic 6.0

Les événements et les exceptions sont des membres de classe qui sont utilisés pour signaler des modifications dans la classe au code client. En général, un événement signale une modification prévue et le code client peut choisir de répondre ou pas. Les exceptions signalent un état d'erreur que le code client ignore à ses risques et périls. Pour plus d'informations, consultez Gestion structurée des exceptions en Visual Basic.

Événements

Visual Basic 6.0

La syntaxe de déclaration d'un événement n'est pas différente de Visual Basic 6.0 :

```
Public Event StateChanged()
```

Visual Basic 2005

La bibliothèque de classes du .NET Framework présente une nouvelle convention de codage des événements. Les événements ont généralement deux paramètres, un paramètre Object nommé *sender* et un paramètre EventArgs nommé *e*. Lorsque l'événement est déclenché, l'objet qui déclenche l'événement passe une référence à lui-même comme paramètre *sender*. Le paramètre **EventArgs** peut être un objet **EventArgs** ou une instance d'une classe qui dérive de **EventArgs**. La classe dérivée contient des informations spécifiques à l'événement qui peuvent être utiles au code client. Voici un exemple :

VB

```
Public Event StateChanged(ByVal sender As Object, ByVal e As EventArgs)
```

Exceptions

Visual Basic 6.0

Les exceptions ne sont pas prises en charge dans Visual Basic 6.0, en revanche, c'est le cas pour le déclenchement d'une erreur. Les erreurs sont interceptées à l'aide d'instructions **On Error GoTo** conjointement avec les instructions étiquetées.

Visual Basic 2005

Le déclenchement d'une erreur est pris en charge dans Visual Basic 2005. Pour plus d'informations, consultez Vue d'ensemble de la gestion non structurée des exceptions.

Des exceptions sont nouvelles dans Visual Basic 2005. La classe Exception sert de classe de base pour toutes les exceptions. (Pour plus d'informations, consultez Classe et propriétés d'exception). Au lieu de créer des numéros d'erreur, comme dans Visual Basic 6.0, le programmeur crée des classes dérivées. Cela peut rendre le code plus lisible. Au lieu des instructions **GoTo** et des instructions étiquetées, la structure en bloc Try...Catch...Finally, instruction (Visual Basic) est utilisée pour répondre aux exceptions. Voici un exemple :

VB

```
Dim big As Integer = Integer.MaxValue
Dim bigger As Integer = Integer.MaxValue
Try
    Dim biggest As Integer = big + bigger
Catch ex As Exception
    MsgBox("Addition did not succeed.")
End Try
```

Pour plus d'informations, consultez Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Suggestions de mise à niveau

Le code qui déclenche des erreurs dans Visual Basic 6.0 est converti en code qui déclenche des erreurs dans Visual Basic 2005. Voici un exemple de code converti qui déclenche une erreur :

VB

```
Private mvarAge As Short
Public Property Age() As Short
    Get
        Age = mvarAge
    End Get
    Set(ByVal Value As Short)
        If Value >= 0 Then
            mvarAge = Value
        Else
            Err.Raise(vbObjectError + 2, Me, _
                "Age must be greater than or equal to zero.")
        End If
    End Set
End Property
```

L'utilisation d'une exception à la place requiert uniquement de remplacer l'appel `Err.Raise` par un Throw, instruction (Visual Basic). Un autre avantage est que le numéro d'erreur peut être éliminé.

Lorsque vous mettez à niveau des événements et des exceptions vers Visual Basic 2005, envisagez de :

- Remplacer les instructions **On Error GoTo** par des instructions **Try**.
- Remplacer **Err.Raise** par **Throw**
- À l'aide des conventions de paramètre des événements (`Public Event StateChanged(ByVal sender As Object, ByVal e As EventArgs)`).
- Créer des exceptions personnalisées et des classes **EventArgs** par héritage.

Voir aussi

Référence

Throw, instruction (Visual Basic)

EventArgs

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

Modifications de l'interface pour les utilisateurs de Visual Basic 6.0

Une interface, telle que définie en Visual Basic, est un type qui définit une liste de membres publics. Un type **Interface** ne peut être ni instancié, ni implémenté.

Interface et classes

L'interface d'une classe représente sa liste de membres publics. L'instruction **Interface**, dans Visual Basic 2005, vous permet de déclarer un type qui définit une liste de membres publics. Lorsqu'une classe inclut l'instruction **Implements**, elle ajoute les membres de l'interface en implémentant chaque membre de l'interface, à la différence de l'héritage de l'implémentation à l'aide du mot clé **Inherits**. Pour l'héritage de l'implémentation, les membres sont hérités de la classe de base et n'ont pas à être implémentés dans la classe dérivée.

Dans le .NET Framework, les interfaces sont souvent utilisées par les classes pour fournir l'assistance à certains services de l'application. À titre d'exemple, le .NET Framework prévoit qu'en cas d'implémentation par une classe de l'interface System.IComparable, le runtime triera des instances de la classe si elles sont contenues dans un ArrayList.

Pour plus d'informations sur l'utilisation des interfaces de Visual Basic 2005, consultez Interfaces en Visual Basic.

Visual Basic 6.0

Dans Visual Basic 6.0, toute classe peut agir comme une interface. Toute classe peut implémenter l'interface d'une autre classe, à l'aide de l'instruction Implements, comme indiqué ci-dessous.

```
' Contents of class BaseClass
Public Sub BaseMethod()
End Sub

' Contents of class DerivedClass
Implements BaseClass
Private Sub BaseClass_BaseMethod()
End Sub
```

Si la présence de code dans les membres individuels de la classe n'est pas requise, elle est possible.

Visual Basic 2005

Visual Basic 2005 fait une nette différence entre les classes et les interfaces. La classe est déclarée dans une instruction **Class** ; l'interface est déclarée dans une instruction **Interface**. Une classe ne peut pas agir comme une interface. Pour agir comme une interface, le type doit être déclaré à l'aide de **Interface**. Les membres d'une interface ne sont pas implémentés. En fait, la syntaxe n'autorise pas des lignes de code telles que `End Sub` dans une définition d'interface. Les membres d'une interface sont implémentés dans la classe qui déclare une interface à l'aide de l'instruction **Implements** :

VB

```
Interface Printable
    Sub Print()
    Property Mode() As Integer
End Interface

Public Class Tree
    Implements Printable

    Private modeValue As Integer
    Public Property Mode() As Integer Implements Printable.Mode
        Get
            Return modeValue
        End Get
        Set(ByVal Value As Integer)
            modeValue = Value
        End Set
    End Property

    Public Sub Print() Implements Printable.Print
```

```
        ' Add code to print a tree.
    End Sub
End Class
```

Suggestions de mise à niveau

La seule solution pour implémenter l'héritage dans Visual Basic 6.0 consiste à utiliser l'instruction **Implements** et une classe de base. Dans les versions plus récentes de Visual Basic, il existe deux types d'héritages : l'héritage de l'implémentation et l'héritage de l'interface. Le type que vous devez sélectionner dépend de votre application. Les points à prendre en compte sont les suivants :

- L'héritage de l'implémentation, à l'aide de l'instruction Inherits, vous permet de créer de nouvelles classes sans écrire davantage de code. Vous n'ajoutez du code que lorsque vous souhaitez modifier le comportement de la classe de base. Pour l'héritage de l'interface, vous devez ajouter du code pour chaque membre de la classe de base.
- Les interfaces peuvent être utilisées lorsque vous souhaitez hériter de plusieurs classes. Visual Basic 2005 prend en charge l'héritage d'une seule classe, mais une classe peut implémenter plusieurs interfaces.
- Le .NET Framework définit plusieurs interfaces qui simplifient des tâches de programmation courantes. **IComparable**, ISerializable et IFormattable en sont des exemples.
- Pour de nombreuses interfaces .NET Framework, le .NET Framework fournit également une classe qui implémente l'interface. La classe Component qui implémente l'interface IComponent en est un exemple. En héritant de **Component** à l'aide de l'instruction Inherits, une classe bénéficie de toutes les fonctionnalités de l'interface **IComponent** sans qu'il soit nécessaire d'écrire du code.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Class, instruction (Visual Basic)
Inherits, instruction

Modifications orientées objet pour les modules en Visual Basic

Les membres partagés sont des membres de classe, tels que les propriétés et procédures, qui sont disponibles pour toutes les instances d'une classe, mais ne sont associés à aucune instance particulière d'une classe. Pour plus d'informations sur les membres partagés, consultez Membres partagés en Visual Basic.

Données d'instance et données partagées

Les données d'instance sont le groupe de données créé lorsque le mot clé **New** est exécuté. En revanche, les données partagées d'une classe sont créées indépendamment de toute exécution du mot clé **New**.

Visual Basic 6.0

Visual Basic 6.0 ne prend pas en charge des données partagées dans les définitions de classe. Pour partager des données entre des instances de classe, une variable globale dans un module est utilisée.

Visual Basic 2005

Les données partagées sont déclarées en ajoutant le modificateur Shared (Visual Basic) à l'instruction de déclaration de variable, comme indiqué.

VB

```
Private Shared totalInstances As Integer
```

Méthodes partagées

Visual Basic 6.0

Visual Basic 6.0 ne prend pas en charge des membres de classe partagés. Des fonctionnalités semblables sont disponibles à partir de modules. L'exécution crée seulement une instance des données dans un module. Les données et méthodes d'un module ont une portée qui correspond à un projet. Voici un exemple d'une bibliothèque implémentée dans un module :

```
Public Sub GetTitle(wordXml As String)
    ' Add code here to find title in Xml string.
End Sub

Public Sub ReplaceTitle(wordXml As String, newTitle As String)
    ' Add code here to replace title.
End Sub
```

Visual Basic 2005

Une approche des modules plus orientée objet utilisera des membres de classe partagés, si les membres sont associés les uns aux autres comme une bibliothèque ou si les membres sont associés à une classe existante. La conception déterminera si tous les membres sont partagés ou si une classe comporte un mélange de membres partagés et membres d'instance. Si la classe comporte uniquement des membres partagés, l'ajout d'un seul constructeur privé empêchera la classe d'être instanciée par code client. L'exemple suivant montre une classe qui ne peut pas être instanciée par le code client et le code client qui appelle l'un des membres.

VB

```
Public Class WordLibrary
    Public Shared Sub GetTitle(ByVal wordXml As String)
        ' Add code here to find title in Xml string.
    End Sub

    Public Shared Sub ReplaceTitle(ByVal wordXml As String, _
        ByVal newTitle As String)
        ' Add code here to replace title.
    End Sub

    Private Sub New()
        ' This prevents instantiation of the class in the client.
    End Sub
```

```
End Class
```

VB

```
' Code that calls the library method.
Sub ChangeTheTitle()
    WordLibrary.ReplaceTitle("Old Title", "New Title")
End Sub
```

L'utilisation d'une classe sur un module de cette bibliothèque offre des avantages, parmi lesquels :

- La bibliothèque a un nom, `WordLibrary`qui est appelé explicitement dans le code.
- Les méthodes sont groupées dans une classe, indiquant leur relation les unes avec les autres.
- L'utilisation d'un constructeur privé indique clairement que cette classe fournit un ensemble de méthodes utilitaires.
- Le code client utilise le nom de la bibliothèque, indiquant clairement l'organisation du projet au lecteur.

Dans le .NET Framework, les méthodes partagées sont généralement utilisées pour fournir des fonctionnalités qui sont associés à une classe, mais ne requièrent pas une instance de la classe. La méthode **Parse** qui est prise en charge par de nombreux types de données dans le .NET Framework en est un exemple. Integer, type de données (Visual Basic) prend en charge une méthode **Parse** qui accepte une chaîne comme paramètre. La méthode **Parse** retourne l'entier représenté par la chaîne. La méthode **Parse** ne peut logiquement être une méthode membre. Il n'y a aucune valeur entière à manipuler, uniquement une chaîne.

Accès à des membres partagés

Visual Basic 6.0

Visual Basic 6.0 ne prend pas en charge des membres de classe partagés.

Visual Basic 2005

Il est possible d'accéder aux membres partagés à l'aide du nom de classe ou du nom de l'instance, comme indiqué.

VB

```
WordLibrary.ReplaceTitle("Old Title", "New Title")
```

Les méthodes partagées ne peuvent pas accéder aux données d'instance à moins qu'une référence à une instance de la classe soit passée à la méthode partagée.

Suggestions relatives à la mise à niveau

Le module `WordLibrary` de Visual Basic 6.0 est mis à niveau vers le code suivant :

VB

```
Module WordLibrary
    Public Sub GetTitle(ByRef wordXml As String)
        ' Add code here to find title in Xml string.
    End Sub

    Public Sub ReplaceTitle(ByRef wordXml As String, _
        ByRef newTitle As String)
        ' Add code here to replace title.
    End Sub
End Module
```

Lors de la mise à niveau du code de module de Visual Basic 6.0 vers Visual Basic 2005, prenez en compte ce qui suit :

- Organisation des méthodes de module et des données dans les classes en tant que données partagées et méthodes.
- Déplacement des méthodes de module dans des classes existantes en tant que méthodes partagées, lorsque la méthode

est associée à la classe.

Voir aussi

Concepts

Membres partagés en Visual Basic

Modifications de la compatibilité binaire pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 met à niveau le mécanisme afin de préserver la compatibilité avec des versions antérieures d'un composant.

Compatibilité binaire

Visual Basic 6.0

Dans Visual Basic 6.0, l'option de **compatibilité binaire** vous permet de conserver automatiquement les identificateurs de classe et d'interface à partir d'une version antérieure d'un composant lors de la compilation d'une nouvelle version. La nouvelle version contient à la fois les anciens et les nouveaux identificateurs, et les applications clientes peuvent continuer à utiliser l'ancienne interface de votre composant sans devoir effectuer de mise à jour.

Visual Basic 2005

Dans Visual Basic 2005, la compatibilité binaire est réalisée à l'aide des attributs. Ceci vous donne un contrôle direct sur les informations situées dans votre composant compilé, telles que les identificateurs de classe et d'interface, les offsets de table virtuelle et les attributs COM appropriés. Puisque vous pouvez sélectionner explicitement des attributs à des fins de conservation, l'option de **compatibilité binaire** n'est pas prise en charge.

Voir aussi

Référence

ComClassAttribute, classe

Concepts

Vue d'ensemble des attributs dans Visual Basic
Attributs utilisés dans Visual Basic
Attributs globaux en Visual Basic
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Comment : convertir une collection personnalisée en collection typée Visual Basic

Dans Visual Basic 6.0, une collection personnalisée limite le contenu d'une collection à une classe particulière. Elle est également appelée collection typée. Visual Basic 2005 fournit plusieurs options pour créer des classes typées à partir des collections personnalisées de Visual Basic 6.0. Cette rubrique contient trois procédures, chacune affichant une option différente.

Cette rubrique suppose que la collection personnalisée de Visual Basic 6.0 a été créée à l'aide de l'**utilitaire Class Builder**. Une telle collection possède les membres suivants :

- **Ajouter** Ajoute une nouvelle instance de la classe personnalisée à la collection.
- **Élément** Retourne une instance de la collection basée sur un index dans la collection.
- **Nombre** Retourne le nombre d'instances dans la collection.
- **Supprimer** Supprime une instance de la collection basée sur un index dans la collection.
- **Énumération** Prend en charge l'énumération à l'aide de la syntaxe **For Each**.

Pour utiliser l'Assistant Mise à niveau pour créer une collection

- Ouvrez le projet Visual Basic 6.0 dans Visual Basic 2005. Pour une classe de collection nommée `Forest` qui contient des instances de la classe `Tree`, le code mis à niveau est indiqué ci-dessous. L'avantage de cette méthode est qu'aucune modification du code n'est requise.

  VB

```
' For this example, the Tree class has no members.
Option Strict Off
Option Explicit On
Friend Class Tree
End Class

Friend Class Forest
  Implements System.Collections.IEnumerable
  'local variable to hold collection
  Private mCol As Collection

  Public Function Add(Optional ByRef sKey As String = "") As Tree
    'create a new object
    Dim objNewMember As Tree
    objNewMember = New Tree
    'set the properties passed into the method
    If Len(sKey) = 0 Then
      mCol.Add(objNewMember)
    Else
      mCol.Add(objNewMember, sKey)
    End If
    'return the object created
    Add = objNewMember
    'UPGRADE_NOTE: Object objNewMember may not be destroyed until it is garbage collec
ted. Click for more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="
vbup1029"'

    objNewMember = Nothing
  End Function

  Default Public ReadOnly Property Item(ByVal vntIndexKey _
    As Object) As Tree
```

```
    Get
      'used when referencing an element in the collection
      'vntIndexKey contains either the Index or Key to the collection,
      'this is why it is declared as a Variant
      'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
      Item = mCol.Item(vntIndexKey)
    End Get
  End Property

  Public ReadOnly Property Count() As Integer
    Get
      'used when retrieving the number of elements in the
      'collection. Syntax: Debug.Print x.Count
      Count = mCol.Count()
    End Get
  End Property

  'UPGRADE_NOTE: NewEnum property was commented out. Click for more: 'ms-help://MS.MSD
NQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1054"'

  'Public ReadOnly Property NewEnum() As stdole.IUnknown
    'Get
      'this property allows you to enumerate
      'this collection with the For...Each syntax
      'NewEnum = mCol._NewEnum
    'End Get
  'End Property

  Public Function GetEnumerator() As System.Collections.IEnumerator _
    Implements System.Collections.IEnumerable.GetEnumerator
    'UPGRADE_TODO: Uncomment and change the following line to return the collection en
umerator. Click for more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyw
ord="vbup1055"'

    'GetEnumerator = mCol.GetEnumerator
  End Function

  Public Sub Remove(ByRef vntIndexKey As Object)
    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mCol.Remove(vntIndexKey)
  End Sub

  'UPGRADE_NOTE: Class_Initialize was upgraded to Class_Initialize_Renamed. Click for
more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1061"'

  Private Sub Class_Initialize_Renamed()
    'creates the collection when this class is created
    mCol = New Collection
  End Sub
  Public Sub New()
    MyBase.New()
    Class_Initialize_Renamed()
  End Sub

  'UPGRADE_NOTE: Class_Terminate was upgraded to Class_Terminate_Renamed. Click for mo
re: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1061"'
```

```
  Private Sub Class_Terminate_Renamed()
    'destroys collection when this class is terminated
    'UPGRADE_NOTE: Object mCol may not be destroyed until it is garbage collected. Cli
ck for more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1029
"'

    mCol = Nothing
  End Sub
  Protected Overrides Sub Finalize()
    Class_Terminate_Renamed()
    MyBase.Finalize()
  End Sub
End Class
```

Le .NET Framework et Visual Basic 2005 proposent plusieurs collections génériques. L'avantage d'utiliser des classes génériques est que très de code est requis pour implémenter la classe.

Pour créer une collection à l'aide de collections génériques

1. Créez la définition de classe. Voici un exemple de classe `Tree` :

   VB

   ```
   Public Class Tree
       Public Species As String
   End Class
   ```

2. Créez une classe de liste générique à partir du .NET Framework. Cette déclaration remplace essentiellement la classe `Forest` tout entière dans la procédure mentionnée ci-dessus intitulée « Pour utiliser l'Assistant Mise à niveau pour créer une collection ».

   VB

   ```
   Dim forest As New System.Collections.Generic.List(Of Tree)
   ```

3. Écrivez du code pour accéder à l'objet Liste. Le code suivant ajoute cinq instances de la classe `Tree` à la collection puis les imprime.

   VB

   ```
   For count As Integer = 1 To 5
       Dim sapling As New Tree
       sapling.Species = "oak"
       Forest.Add(sapling)
   Next

   For Each sapling As Tree In Forest
       MsgBox(sapling.Species)
   Next
   ```

Le .NET Framework inclut la classe CollectionBase. Les collections typées sont créées en héritant de **CollectionBase**. Cette méthode utilise moins de code que la méthode de l'Assistant Mise à niveau, mais plus que la solution générique. Elle plus flexible que la solution générique, parce que le programmeur peut ajouter des membres supplémentaires à la classe de collection.

Pour créer une collection à partir de la classe CollectionBase

1. Créez la définition de classe. Voici un exemple de classe `Tree` :

   VB

   ```
   Public Class Tree
       Public Species As String
   End Class
   ```

2. Créez une classe qui hérite de la classe **CollectionBase**. Ajoutez, au minimum, une méthode `Add` et une propriété `Item`. Elles rendent la classe de collection fortement typée.

   VB

   ```
   Class TreeCollection
     Inherits System.Collections.CollectionBase

     Public Sub Add(ByVal value As Tree)
       Me.List.Add(value)
     End Sub

     Default Public Property Item(ByVal index As Integer) As Tree
         Get
             Return CType(Me.List(index), Tree)
         End Get
         Set(ByVal value As Tree)
             If index <= Me.Count - 1 Then
                 Me.List(index) = value
             Else
                 Throw New IndexOutOfRangeException()
             End If
         End Set
     End Property
   End Class
   ```

3. Écrivez du code pour accéder à l'objet Liste. Le code suivant ajoute cinq instances de la classe `Tree` à la collection puis les imprime.

   VB

   ```
   Dim forest As New TreeCollection
   ```

   VB

   ```
   For count As Integer = 1 To 5
       Dim sapling As New Tree
       sapling.Species = "oak"
       Forest.Add(sapling)
   Next

   For Each sapling As Tree In Forest
       MsgBox(sapling.Species)
   Next
   ```

Voir aussi

Référence
List
CollectionBase
Ienumerable

Comment : convertir un type défini par l'utilisateur en structure Visual Basic

L'instruction **Type** de Visual Basic 6.0 est mise à niveau vers l'instruction **Structure** de Visual Basic 2005. Deux autres options de mise à niveau du type sont également abordées dans cette rubrique : la conversion en une classe et la conversion en une interface.

Le type Visual Basic 6.0 suivant est abordé dans les procédures qui suivent :

```
Type Customer
    Dim FirstName As String
    Dim LastName As String
    Dim Id As Long
End Type

Private Function FormatFullName(aCustomer As Customer) As String
    FormatFullName = aCustomer.FirstName & " " & aCustomer.LastName
End Function
```

L'**Assistant Mise à niveau** met automatiquement à niveau un type Visual Basic 6.0 vers une structure Visual Basic 2005. **Structure** est un type qui prend en charge les membres, tels que les méthodes et les propriétés.

Pour mettre à niveau un type vers une structure

1. Exécutez l'**Assistant Mise à niveau**.

   Remarque

   L'**Assistant Mise à niveau** est un outil utilisé pour mettre à niveau une application Visual Basic 6.0 en Visual Basic 2005. Il est lancé automatiquement à l'ouverture d'un projet Visual Basic 6.0 dans Visual Basic 2005. Pour plus d'informations, consultez Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic.

2. Si vous voulez tirer parti d'autres fonctionnalités du type **Structure**, recherchez des méthodes qui sont plus adaptées à la structure. Dans cet exemple, la méthode `FormatFullName` pourrait être convertie en une méthode `FormatFullName` dans la structure :

   VB

   ```
   Structure Customer
       Dim FirstName As String
       Dim LastName As String
       Dim Id As Integer

       Public Function FormatFullName() As String
           Return FirstName & " " & LastName
       End Function
   End Structure
   ```

L'**Assistant Mise à niveau** mettra automatiquement à niveau un type Visual Basic 6.0 vers une structure Visual Basic 2005. Le type **Class** prend en charge les mêmes membres qu'un **Structure**, mais c'est un type référence. Une classe peut servir de classe de base, alors qu'une structure ne le peut pas.

Pour mettre à niveau vers une classe

1. Exécutez l'**Assistant Mise à niveau**.
2. Remplacez `Structure` par `Class`.
3. Si vous voulez tirer parti d'autres fonctionnalités du type **Class**, recherchez des méthodes qui sont plus adaptées à la

structure. Le code suivant illustre l'incorporation de `FormatFullName` dans la classe, ainsi que la propriété `Id` .

VB

```
Class Customer
    Dim FirstName As String
    Dim LastName As String
    Private idValue As Integer

    Public Property Id() As Integer
        Get
            Return idValue
        End Get
        Set(ByVal Value As Integer)
            idValue = Value
        End Set
    End Property

    Public Function FormatFullName() As String
        Return FirstName & " " & LastName
    End Function
End Class
```

Une troisième option est la conversion de la structure en une interface.

Pour mettre à niveau vers une interface

1. Exécutez l'**Assistant Mise à niveau**.
2. Remplacez `Structure` par `Interface`.
3. Remplacez les variables par les déclarations de propriété. N'incluez pas l'implémentation.
4. Ajoutez les méthodes, mais supprimez l'implémentation.
5. Le code se présente comme suit :

   VB

   ```
   Interface Customer
       Property FirstName() As String
       Property LastName() As String
       Property Id() As String

       Function FormatFullName() As String
   End Interface
   ```

6. L'interface peut ensuite être implémentée par une autre classe. La définition est indiquée ci-dessous.

   VB

   ```
   Class NewCustomer
       Implements Customer

       Public Property FirstName() As String Implements Customer.FirstName
           Get
               ' Add code here.
           End Get
           Set(ByVal Value As String)
               ' Add code here.
           End Set
   ```

```
    End Property

    Public Property Id() As String Implements Customer.Id
        Get
            ' Add code here.
        End Get
        Set(ByVal Value As String)
            ' Add code here.
        End Set
    End Property

    Public Property LastName() As String Implements Customer.LastName
        Get
            ' Add code here.
        End Get
        Set(ByVal Value As String)
            ' Add code here.
        End Set
    End Property

    Public Function FormatFullName() As String _
        Implements Customer.FormatFullName
        ' Add code here.
    End Function
End Class
```

Voir aussi

Tâches

Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic

Référence

Structure, instruction

Class, instruction (Visual Basic)

Interface, instruction (Visual Basic)

Comment : convertir du code Implements en nouvelle forme d'héritage

Visual Basic 6.0 utilise l'instruction **Implements** pour l'héritage de l'interface. Des versions plus récentes de Visual Basic proposent deux types d'héritages, à l'aide de Implements, instruction et de Inherits, instruction. Cette rubrique illustre comment convertir du code à l'aide de ces deux options.

Le code Visual Basic 6.0 suivant est pris en compte dans cette page :

```
' Account class.
Private mvarBalance As Currency
Private mvarOwner As String

Public Property Let Owner(ByVal vData As String)
    mvarOwner = vData
End Property

Public Property Get Owner() As String
    Owner = mvarOwner
End Property

Public Function Deposit(ByVal amount As Currency) As Currency
End Function

' Savings class.
Private mvarBalance As Currency
Private mvarOwner As String 'local copy

Public Property Let Owner(ByVal vData As String)
    mvarOwner = vData
End Property

Public Property Get Owner() As String
    Owner = mvarOwner
End Property

Public Function Deposit(ByVal amount As Currency) As Currency
End Function
```

Pour convertir à l'aide d'une interface commune

1. Exécutez l'**Assistant Mise à niveau**.

   Remarque

   L'**Assistant Mise à niveau** est un outil utilisé pour mettre à niveau une application Visual Basic 6.0 en Visual Basic 2005. Il est lancé automatiquement à l'ouverture d'un projet Visual Basic 6.0 dans Visual Basic 2005. Pour plus d'informations, consultez Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic.

   L'Assistant convertira la classe `Account` en une interface (`_Account`) et créera deux classes qui implémenteront l'interface (`Account` et `Savings`.)

   VB

```
Interface _Account
    Property Owner() As String
    Function Deposit(ByVal amount As Decimal) As Decimal
End Interface
Friend Class Account
    Implements _Account
    Private mvarBalance As Decimal
```

```
    Private mvarOwner As String

    Public Property Owner() As String Implements _Account.Owner
        Get
            Owner = mvarOwner
        End Get
        Set(ByVal Value As String)
            mvarOwner = Value
        End Set
    End Property

    Public Function Deposit(ByVal amount As Decimal) _
        As Decimal Implements _Account.Deposit
    End Function
End Class

Friend Class Savings
    Implements _Account

    Private Function Account_Deposit(ByVal amount As Decimal) _
        As Decimal Implements _Account.Deposit
    End Function

    Private Property Account_Owner() As String _
        Implements _Account.Owner
        Get
        End Get
        Set(ByVal Value As String)
        End Set
    End Property

    Public Function AddInterest() As Object
    End Function
End Class
```

2. Le code permettant de créer et utiliser une instance du compte `Savings` pourrait ressembler à ceci.

   VB

```
Dim savings As _Account
savings = New Savings
savings.Owner = "Jeff Pike"
```

Si vous souhaitez que `Account` soit la classe de base, vous pouvez modifier la sortie de l'Assistant.

Pour convertir avec Interface et Implémente

1. Exécutez l'**Assistant Mise à niveau**.
2. Supprimez la classe `Account`.
3. Remplacez le nom de la classe `_Account` par `Account`.
4. Dans la classe `Savings`, remplacez `_Account` par `Account`.

   VB

```
Friend Class Savings
    Implements Account
```

```
    Private Function Account_Deposit(ByVal amount As Decimal) _
        As Decimal Implements Account.Deposit
    End Function

    Private Property Account_Owner() As String Implements Account.Owner
        Get
        End Get
        Set(ByVal Value As String)
        End Set
    End Property

    Public Function AddInterest() As Object
    End Function
End Class
```

5. Le code permettant de créer et utiliser une instance du compte Savings pourrait ressembler à ceci.

   VB

```
Dim savings As Account
savings = New Savings
savings.Owner = "Jeff Pike"
```

Si vous souhaitez que la classe `Account` soit une classe de base, vous pouvez modifier la sortie de l'Assistant.

Pour convertir avec Classe et Hérite

1. Exécutez l'**Assistant Mise à niveau**.
2. Supprimez l'interface `_Account`.
3. Dans la classe `Account`, supprimez les références au `_Account` dans les instructions **Implements**.

   VB

```
Friend Class Account
    Private mvarBalance As Decimal
    Private mvarOwner As String

    Public Property Owner() As String
        Get
            Owner = mvarOwner
        End Get
        Set(ByVal Value As String)
            mvarOwner = Value
        End Set
    End Property

    Public Function Deposit(ByVal amount As Decimal) As Decimal
    End Function
End Class
```

4. Dans la classe `Savings`, remplacez `Implements _Account` par `Inherits Account`.
5. Dans la classe `Savings`, supprimez les membres `Owner` et `Deposit`.

   VB

```
Friend Class Savings
    Inherits Account

    Public Function AddInterest() As Object
    End Function
End Class
```

6. Le code permettant de créer et utiliser une instance du compte `Savings` pourrait ressembler à ceci.

   VB

```
Dim savings As Account
savings = New Savings
savings.Owner = "Jeff Pike"
```

Voir aussi

Tâches

Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic

Référence

Class, instruction (Visual Basic)
Inherits, instruction
Interface, instruction (Visual Basic)
Implements, instruction

Autres ressources

Héritage en Visual Basic
Interfaces en Visual Basic

Programmation Web pour les utilisateurs de Visual Basic 6.0

Visual Basic 6.0 prend en charge la programmation Web directement. Dans Visual Basic 2005, la programmation Web s'effectue à l'aide de Visual Web Developer et du langage Visual Basic.

Différences conceptuelles

Visual Basic 6.0 offre plusieurs fonctionnalités prenant en charge la programmation Web : applications IIS (Internet Information Services) (classes Web), applications DHTML, documents ActiveX et contrôles ActiveX pouvant être téléchargés sur des pages Web.

Dans Visual Basic 2005, la programmation Web n'est plus prise en charge directement. À la place, vous pouvez utiliser Visual Web Developer et le langage Visual Basic pour créer des sites Web ASP.NET, des services Web ASP.NET, et beaucoup plus. Vous pouvez créer des contrôles Web dans Visual Basic 2005 qui peuvent être utilisés dans Visual Web Developer, et vous pouvez consommer des services Web ASP.NET d'une application ou d'un composant Visual Basic 2005.

ASP et ASP.NET

Dans Visual Basic 6.0, les applications IIS (Internet Information Services) utilisent le modèle ASP (Active Server Pages) pour créer des applications qui s'exécutent sur IIS. La technologie ASP.NET vous permet de créer un site Web à l'aide de pages Web Forms ; les applications de site Web ASP.NET sont également hébergées sur IIS.

Applications DHTML

Les applications DHTML de Visual Basic 6.0 utilisent le modèle objet DHTML et le code Visual Basic pour créer des applications capables de répondre aux actions exécutées par l'utilisateur dans un navigateur. Les Web Forms Visual Web Developer se développent à partir du modèle DHTML, fournissant des capacités d'interface utilisateur dynamiques plus riches ainsi que la validation côté client.

Documents et contrôles ActiveX

Les documents ActiveX de Visual Basic 6.0 ne sont pas pris en charge dans Visual Basic 2005. Vous pouvez toujours interagir avec les documents ActiveX depuis vos applications Web Visual Web Developer. Toutefois, leur développement doit être maintenu dans Visual Basic 6.0.

Visual Basic 2005 vous permet également de créer des contrôles ActiveX qui peuvent être téléchargés vers des pages Web, ou d'utiliser des contrôles ActiveX existants dans vos applications Visual Basic 2005.

Voir aussi

Concepts

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Autres ressources

Visual Web Developer
Aide pour les utilisateurs de Visual Basic 6.0
Espaces de noms Web dans Visual Studio
Programmation du Web avec les services Web XML

WebClasses pour les utilisateurs de Visual Basic 6.0

Pour l'essentiel, les concepts et les techniques de programmation d'un site Web ASP.NET dans Visual Basic 2005 sont très différents des concepts et des techniques de création de classes Web dans Visual Basic 6.0.

Différences conceptuelles

Dans Visual Basic 6.0, les projets WebClass (également appelés projets d'application IIS) sont utilisés pour créer des applications Web basées sur la technologie ASP (Active Server Page).

Dans Visual Basic 2005, les projets de site Web ASP.NET sont utilisés pour créer des applications Web basées sur la technologie ASP.NET, plus récente. Pour plus d'informations, consultez
Visite guidée de la création de sites Web dans Visual Web Developer.

Gestion des états

Dans Visual Basic 6.0, les projets WebClass disposent d'une propriété **StateManagement** qui permet de conserver une instance de WebClass active entre les demandes. Pour ce faire, la propriété **StateManagement** doit se voir attribuer la valeur **2** ou **wcRetainInstance** au moment du design. Une méthode **ReleaseInstance** peut également être utilisée pour terminer une instance.

Dans Visual Basic 2005, les applications Web ASP.NET n'ont pas de propriété **StateManagement**. Le modèle de gestion de l'état de l'application est très différent, et tout le code relatif à la gestion de l'état doit être remplacé. Pour plus d'informations, consultez Gestion d'état ASP.NET.

Mettre à niveau des remarques

Lorsqu'un projet WebClass Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, il est converti en un projet de site Web ASP.NET.

Des déclarations sont ajoutées à votre projet : une pour le projet WebClass et une pour chaque objet WebItem et Template du projet WebClass. Une procédure événementielle **Page_Load** est ajoutée au projet, en créant d'abord un objet WebClass, puis des objets WebItem pour chacun des objets WebItem et Template associés au projet WebClass Visual Basic 6.0. Enfin, dans la procédure événementielle **Page_Load**, vous verrez un appel au runtime WebClass Compatibility : **WebClass.ProcessEvents**. Cet appel permet au runtime de générer le rendu du WebItem spécifié dans l'URL de la demande. Ce code est le seul nouveau code ajouté à votre projet mis à niveau et ne sert qu'à émuler le comportement sous-jacent du runtime WebClass Visual Basic 6.0.

Les procédures **Function** et **Sub** dans votre code Visual Basic 6.0 (par exemple, **ProcessTags** ou **Respond**) auront leur portée changée de **Private** en **Public** pour permettre au runtime WebClass Compatibility de les exécuter.

Certains événements WebClass Visual Basic 6.0 ne sont pas pris en charge dans ASP.NET (par exemple, **Initialize**, **BeginRequest**, **EndRequest** et **Terminate**). Ces procédures événementielles seront mises à niveau, mais elles ne seront pas appelées lors de l'exécution. Après la mise à niveau, vous devrez convertir tout code présent dans ces événements en son équivalent ASP.NET.

Remarque

Il existe un certain nombre de différences en termes de propriétés, de méthodes et d'événements entre les objets ASP et ASP.NET. Avant de procéder à la mise à niveau d'un projet WebClass vers ASP.NET, vous devez comprendre ces différences afin de savoir comment modifier votre code. Pour plus d'informations, consultez
Visite guidée de la création de sites Web dans Visual Web Developer.

Voir aussi

Référence

L'événement WebClass n'est pas pris en charge

Concepts

Mise à niveau de projets WebClass vers Visual Basic 2005

Autres ressources

Introduction aux pages Web ASP.NET (Visual Studio)

Accès aux données pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, l'accès aux données s'effectue à l'aide d'ADO (ActiveX Data Objects). Dans Visual Basic 2005, l'accès aux données s'effectue à l'aide de ADO.NET, qui est une partie du .NET Framework. Il existe plusieurs différences, à la fois conceptuellement et en termes de tâches, entre les deux technologies. Pour plus d'informations sur les différences conceptuelles entre ADO et ADO.NET, consultez Comparaison entre ADO.NET et ADO.

Remarque

Visual Basic 6.0 assure également la compatibilité descendante pour les objets RDO (Remote Data Objects) et DAO (Data Access Objects). Ces technologies sont prises en charge dans Visual Basic 2005 comme objets COM uniquement ; vous ne pouvez pas mettre à niveau vers Visual Basic 2005 des applications utilisant la liaison de données RDO ou DAO. Pour plus d'informations, consultez Mise à niveau d'applications créées dans des versions précédentes de Visual Basic.

Dans Visual Basic 6.0, il existe deux méthodes communes d'implémentation de l'accès aux données dans une application : au moment du design, en créant une liaison à un ADODC (contrôle de données ADO) ou en utilisant un environnement de données, ou au moment de l'exécution, en créant et en interagissant par programme avec les objets **Recordset**.

Dans Visual Basic 2005, deux méthodes semblables permettent d'implémenter l'accès aux données : au moment du design en utilisant des adaptateurs de données et des groupes de données, ou au moment de l'exécution en ajoutant du code pour créer des adaptateurs de données et des groupes de données par programme. Pour plus d'informations, consultez Création d'adaptateurs de données.

Dans Visual Basic 6.0, la liaison de données s'effectue par définition des propriétés connexes à liaison d'un contrôle : **DataChanged**, **DataField**, **DataFormat**, **DataMember** et **DataSource**. Dans la plupart des cas, la propriété d'affichage d'un contrôle (par exemple, la propriété **Text** d'un contrôle **TextBox**) est liée à un champ dans une source de données.

Dans Visual Basic 2005, la définition d'une liaison de données est beaucoup plus large ; vous pouvez lier n'importe quelle propriété de n'importe quel contrôle à toute structure contenant des données. La liaison dans les Windows Forms s'effectue au moyen d'une propriété **DataBindings** qui contient une collection d'objets **Binding** pour le contrôle. Pour plus d'informations, consultez Liaison de données et Windows Forms.

Remarque

La liaison de données ADO est prise en charge pour assurer la compatibilité descendante avec le runtime de données de compatibilité de Microsoft Visual Basic 2005.

Dans Visual Basic 6.0, les formulaires liés aux données sont créés automatiquement par glisser-déplacer des éléments de base de données d'un environnement de données sur un formulaire.

Voir aussi

Autres ressources

Création d'applications de données clientes
Connexion aux données avec l'Explorateur de serveurs/Explorateur de bases de données
Visual Database Tools
Aide pour les utilisateurs de Visual Basic 6.0
Espaces de noms de données dans Visual Studio

Création de composants pour les utilisateurs de Visual Basic 6.0

Bien que Visual Basic 6.0 et Visual Basic 2005 fournissent tous deux des fonctions de création de composants, les différences dans les modes de création des composants sont importantes.

Différences conceptuelles

Dans Visual Basic 6.0, la création de composants est synonyme de création de composants COM : contrôles ActiveX, dll ActiveX et exe ActiveX pouvant être utilisés dans des applications COM.

Dans Visual Basic 2005, les composants reposent sur le .NET Framework ; vous créez des composants qui peuvent être utilisés dans les applications construites à l'aide du .NET Framework. Les composants créés avec Visual Basic 2005 sont fondés sur l'héritage : tout composant dérive de la classe de base de **Component** ou de **Control**. Pour plus d'informations, consultez Classes de composant.

Composants multithread

Visual Basic 6.0 fournit deux modèles de threading : les composants peuvent être mono-thread ou à threads cloisonnés.

Visual Basic 2005 prend en charge les composants multithread véritables. Pour plus d'informations, consultez Multithreading dans les composants.

Instanciation de composant

Dans Visual Basic 6.0, la propriété **Instancing** d'une classe contrôle le niveau d'accès et la manière avec laquelle un composant peut être créé.

Dans Visual Basic 2005, l'instanciation est contrôlée par la définition du modificateur d'accès et du niveau d'accès dans le constructeur d'une classe de composants. Pour plus d'informations, consultez Modifications des instances de composants dans Visual Basic.

Compatibilité binaire et suivi des versions

Dans Visual Basic 6.0, la compatibilité binaire des composants est déterminante pour éviter les conflits de version ; elle est contrôlée par la propriété **Version Compatibility** du projet du composant.

Dans Visual Basic 2005, le versioning est intégré aux assemblys de composants ; vous n'avez plus à définir la compatibilité binaire. Pour plus d'informations, consultez Programmation à l'aide de composants.

Contrôles ActiveX

Dans Visual Basic 6.0, vous pouvez créer des contrôles ActiveX (également dénommés contrôles utilisateur) qui peuvent être utilisés dans les applications Windows.

Dans Visual Basic 2005, vous pouvez créer des contrôles utilisateur pour les Windows Forms, ou vous pouvez hériter des contrôles existants pour ajouter des fonctionnalités. Pour plus d'informations, consultez Développement de contrôles Windows Forms au moment du design.

Messagerie

Dans Visual Basic 6.0, vous pouvez créer des composants qui utilisent Microsoft Message Queue (MSMQ) pour transmettre les informations entre les applications.

Dans Visual Basic 2005, la messagerie est construite dans le .NET Framework. Pour plus d'informations, consultez Utilisation de composants de messagerie.

Transactions

Dans Visual Basic 6.0, vous pouvez créer des composants MTS qui fonctionnent avec Microsoft Transaction Server (MTS) pour participer aux transactions.

Avec Visual Basic 2005, les transactions sont contrôlées par le .NET Framework en ajoutant un composant transactionnel dans votre projet ou en ajoutant un attribut **Transaction** à une classe existante.

Voir aussi

Concepts

Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0
Héritage pour les utilisateurs de Visual Basic 6.0

Autres ressources

Création de composants
Aide pour les utilisateurs de Visual Basic 6.0
Espace de noms de modèles de composants dans Visual Studio

Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0

Visual Basic 6.0 et Visual Basic 2005 vous permettent tous deux de créer des contrôles. Il existe toutefois plusieurs différences dont vous devez avoir connaissance.

Différences conceptuelles

Dans Visual Basic 6.0, les projets **UserControl** (également appelés projets ActiveX Control) sont utilisés pour créer des contrôles ActiveX. Une fois compilés, les objets UserControl peuvent être hébergés dans n'importe quel conteneur prenant en charge ActiveX, notamment les formulaires Visual Basic 6.0 et Internet Explorer.

Dans Visual Basic 2005, les projets de bibliothèque de contrôles Windows sont utilisés pour créer des contrôles qui peuvent être hébergés dans des applications Windows Forms.

Comportement au moment du design

Dans Visual Basic 6.0, avant d'ajouter un objet **UserControl** dans un formulaire au moment du design, vous devez fermer la **fenêtre de conception du contrôle UserControl**. Cette étape compile (génère) automatiquement le contrôle. Lorsque des modifications sont apportées par la suite à l'objet **UserControl** , elles ne sont répercutés dans l'instance de **UserControl** sur le formulaire avant l'exécution de l'application. Si elle est ouverte, la **fenêtre de conception du contrôle UserControl** se ferme automatiquement.

Dans Visual Basic 2005, un objet **UserControl** peut être ajouté à un formulaire sans fermer la **fenêtre de conception du contrôle UserControl** ; cependant, le contrôle UserControl doit être généré avant d'être ajouté à un formulaire. La simple fermeture de la **fenêtre de conception du contrôle UserControl** ne suffit pas à générer le contrôle. Toute modification apportée ultérieurement au contrôle utilisateur n'est pas reflétée sur le Concepteur de formulaires avant que l'objet **UserControl** soit régénéré. Un objet **UserControl** est régénéré automatiquement chaque fois vous déboguez votre application de test.

Rendre des valeurs persistantes

Dans Visual Basic 6.0, les événements **ReadProperties** et **WriteProperties** sont utilisés pour récupérer ou enregistrer les valeurs d'un objet **UserControl** dans un objet **PropertyBag**.

Dans Visual Basic 2005, l'objet **PropertyBag** n'est plus pris en charge et les événements **ReadProperties** et **WriteProperties** n'existent plus. À présent, la sérialisation permet d'enregistrer et de récupérer des propriétés au format binaire ou SOAP. Pour plus d'informations, consultez Procédure pas à pas : persistance d'un objet en Visual Basic

Mettre à niveau des remarques

Lorsqu'un projet **UserControl** Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, il est mis à niveau vers une bibliothèque de contrôles Windows ; le contrôle résultant ne peut plus être hébergé dans Internet Explorer.

Voir aussi

Tâches

Procédure pas à pas : persistance d'un objet en Visual Basic

Concepts

Équivalents PropertyBag pour les utilisateurs de Visual Basic 6.0

Autres ressources

Développement de contrôles Windows Forms au moment du design

Équivalents PropertyBag pour les utilisateurs de Visual Basic 6.0

L'objet **PropertyBag** de Visual Basic 6.0 est remplacé par l'objet **My.Settings** dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6, l'objet **PropertyBag** est utilisé pour rendre les données d'un objet persistantes entre des instances, ce qui vous permet de stocker des valeurs et de les récupérér lors de la prochaine instanciation de l'objet. Bien que vous puissiez définir les propriétés d'un objet à une valeur par défaut au moment du design, toute valeur entrée au moment de l'exécution sont perdues lorsque l'objet est détruit. L'objet **PropertyBag** de Visual Basic 6.0 permet la persistance d'une copie de l'objet dans un format binaire, ainsi que sa récupération en vue d'une utilisation ultérieure. Par exemple, une application permettant de calculer des prêts peut utiliser un objet **PropertyBag** pour rendre un taux d'intérêt persistant d'une instance à une autre, afin d'éviter d'avoir à le saisir à chaque exécution de l'application.

Dans Visual Basic 2005, l'objet **PropertyBag** n'existe plus, mais vous pouvez quand même faire persister les données d'un objet à l'aide de l'objet **My.Settings**. L'objet **My.Settings** fournit l'accès aux paramètres d'un composant ou d'une application, et vous permet de stocker et de récupérer dynamiquement des paramètres de propriété et d'autres informations au moment du design et de l'exécution. Pour plus d'informations, consultez Gestion des paramètres de l'application.

Remarque

L'objet **My.Settings** peut uniquement être utilisé pour rendre persistantes des données dans les types définis par le .NET Fra mework. Pour les types de données personnalisés, vous pouvez utiliser la sérialisation pour rendre les données persistantes. Pour plus d'informations, consultez Procédure pas à pas : persistance d'un objet en Visual Basic.

Voir aussi

Concepts

Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0

Windows Forms pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0 (et dans toutes les versions précédentes de Visual Basic), les applications Windows étaient créées à l'aide de formulaires faisant partie intégrante du runtime Visual Basic. Dans Visual Basic 2005, vous créez les applications Windows à l'aide de Windows Forms fournis par le .NET Framework.

Windows Forms et les contrôles Windows Forms offrent une architecture plus complète et plus extensible pour le développement d'applications Windows. Toutefois, il existe certaines différences entre les Windows Forms et les contrôles, et les formulaires et contrôles Visual Basic 6.0. Les rubriques suivantes vous aideront à découvrir ces différences.

Dans cette section

Tâches des formulaires pour les utilisateurs de Visual Basic 6.0

Décrit les différences de tâches courantes lors de l'utilisation de formulaires.

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers les rubriques expliquant les différences entre les contrôles et les objets Windows Forms et Visual Basic 6.0.

Modifications des propriétés, méthodes et événements des contrôles pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent les différences entre les propriétés, les méthodes et les événements de formulaires, contrôles et autres objets.

Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0

Décrit les différences relatives aux tableaux de contrôles entre Visual Basic 6.0 et Visual Basic 2005.

Positionnement de formulaire pour les utilisateurs de Visual Basic 6.0

Décrit les différences de définition de l'emplacement d'un formulaire à l'écran entre Visual Basic 6.0 et Visual Basic 2005.

Objet Menu pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière de création de menus entre Visual Basic 6.0 et Visual Basic 2005.

MDI pour les utilisateurs de Visual Basic 6.0

Décrit les différences relatives aux applications MDI (Multiple-Document Interface) entre Visual Basic 6.0 et Visual Basic 2005.

Modifications des boîtes de dialogue pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière de création de boîtes de dialogue entre Visual Basic 6.0 et Visual Basic 2005.

Comportement des couleurs pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière de gestion des couleurs entre Visual Basic 6.0 et Visual Basic 2005.

Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière d'implémentation d'aide entre Visual Basic 6.0 et Visual Basic 2005.

Modifications de l'impression pour les utilisateurs de Visual Basic 6.0

Décrit les différences de prise en charge de l'impression pour les applications entre Visual Basic 6.0 et Visual Basic 2005.

Rubriques connexes

Événements et gestion des événements pour les utilisateurs Visual Basic 6.0

Décrit les différences en matière de développement de gestionnaires d'événements entre Visual Basic 6.0 et Visual Basic 2005.

Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière d'implémentation d'opérations de glisser-déplacer entre Visual Basic 6.0 et Visual Basic 2005.

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Décrit les différences en matière de méthodes graphiques entre Visual Basic 6.0 et Visual Basic 2005.

Vue d'ensemble des Windows Forms

Présente les concepts fondamentaux des formulaires.

Nouveautés dans les Windows Forms et les contrôles Windows Forms

Décrit les nouvelles fonctionnalités des Windows Forms.

Espaces de noms d'applications Windows dans Visual Studio

Présente les espaces de noms liés aux Windows Forms.

Aide pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers d'autres rubriques décrivant les différences entre Visual Basic 6.0 et Visual Basic 2005.

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, de nouveaux contrôles et objets remplacent les contrôles, objets et collections intrinsèques de Visual Basic 6.0. Dans certains cas, les noms ont changé. Dans de nombreux cas, les propriétés, méthodes et événements sont différents.

Dans cette section

Les rubriques suivantes présentent les correspondances entre les contrôles et objets de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Objet App pour les utilisateurs de Visual Basic 6.0

Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0

Objet Clipboard pour les utilisateurs de Visual Basic 6.0

Contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Contrôle CommandButton pour les utilisateurs de Visual Basic 6.0

Contrôle CommonDialog pour les utilisateurs de Visual Basic 6.0

Collection Controls pour les utilisateurs de Visual Basic 6.0

Contrôle de données pour les utilisateurs de Visual Basic 6.0

Contrôle DataGrid pour les utilisateurs de Visual Basic 6.0

Contrôle DirListBox pour les utilisateurs de Visual Basic 6.0

Contrôle DriveListBox pour les utilisateurs de Visual Basic 6.0

Contrôle FileListBox pour les utilisateurs de Visual Basic 6.0

Objet Font pour les utilisateurs de Visual Basic 6.0

Objet Form pour les utilisateurs de Visual Basic 6.0

Collection Forms pour les utilisateurs de Visual Basic 6.0

Contrôle Frame pour les utilisateurs de Visual Basic 6.0

Contrôle HScrollBar pour les utilisateurs de Visual Basic 6.0

Contrôle Image pour les utilisateurs de Visual Basic 6.0

Contrôle Label pour les utilisateurs de Visual Basic 6.0

Contrôle Line pour les utilisateurs de Visual Basic 6.0

Contrôle ListBox pour les utilisateurs de Visual Basic 6.0

Contrôle ListView pour les utilisateurs de Visual Basic 6.0

Contrôle MaskEdBox pour les utilisateurs de Visual Basic 6.0

Objet MDIForm pour les utilisateurs de Visual Basic 6.0

Objet Menu pour les utilisateurs de Visual Basic 6.0

Contrôle conteneur OLE pour les utilisateurs de Visual Basic 6.0

Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0

Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0

Objet imprimante pour les utilisateurs de Visual Basic 6.0

Collection d'imprimantes pour les utilisateurs de Visual Basic 6.0

Contrôle ProgressBar pour les utilisateurs de Visual Basic 6.0

Contrôle de données RDO (Remote Data Objects) pour les utilisateurs de Visual Basic 6.0

Contrôle RichTextBox pour les utilisateurs de Visual Basic 6.0

Objet Screen pour les utilisateurs de Visual Basic 6.0

Contrôle Shape pour les utilisateurs de Visual Basic 6.0

Contrôle StatusBar pour les utilisateurs de Visual Basic 6.0

Contrôle TextBox pour les utilisateurs de Visual Basic 6.0

Contrôle ToolBar pour les utilisateurs de Visual Basic 6.0

Contrôle Timer pour les utilisateurs de Visual Basic 6.0

Contrôle TreeView pour les utilisateurs de Visual Basic 6.0

Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0

Objet VBControlExtender pour les utilisateurs de Visual Basic 6.0

Contrôle VScrollBar pour les utilisateurs de Visual Basic 6.0

Contrôle WebBrowser pour les utilisateurs de Visual Basic 6.0

Rubriques connexes

Contrôles Windows Forms

Fournit des liens vers des rubriques sur les contrôles conçus spécifiquement pour fonctionner avec les Windows Forms.

Modifications des propriétés, méthodes et événements des contrôles pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers des rubriques qui décrivent les différences entre les propriétés, les méthodes et les événements de formulaires, contrôles et autres objets.

Comparaison des contrôles et des objets programmables dans différents langages et dans différentes bibliothèques

Compare différents contrôles et objets.

Nouveautés dans les Windows Forms et les contrôles Windows Forms

Décrit les nouvelles fonctionnalités des Windows Forms.

Objet App pour les utilisateurs de Visual Basic 6.0

Cette rubrique compare l'objet **App** dans Visual Basic 6.0 avec son équivalent dans Visual Basic 2005.

Dans Visual Basic 6.0, **App** est un objet global qui sert à définir et à récupérer des informations au sujet de l'application. Il n'existe aucun équivalent direct pour l'objet **App** dans Visual Basic 2005. Toutefois, la plupart des propriétés, méthodes et événements peuvent être mappés aux membres équivalents du .NET Framework.

Différences conceptuelles

Version Information, propriétés

Les propriétés **Version Information** de l'objet **App** sont remplacées par les attributs **Assembly** dans Visual Basic 2005. Les propriétés **Version Information** sont définies dans la boîte de dialogue **Propriétés du projet**. Les attributs **Assembly** sont définis dans la boîte de dialogue **Informations de l'assembly** qui est disponible en cliquant sur le bouton **Informations de l'application** sur l'onglet **Application** du **Concepteur de projets**.

Remarque
Les propriétés **Version Information** sont stockées dans le fichier projet (.vbp) que vous pouvez modifier à l'aide d'un éditeur de texte tel que le Bloc-notes. Les attributs **Assembly** sont stockés dans le fichier AssemblyInfo.vb et peuvent être modifiés dans l'**éditeur de code**.

File Description, propriété

Dans Visual Basic 6.0, l'objet **App** a les propriétés **FileDescription** et **Title** ; **FileDescription** détermine le texte de description qui apparaît dans l'**Explorateur Windows**.

Dans Visual Basic 2005, il n'y a aucun attribut **FileDescription**. À la place, l'attribut **Title** détermine le texte descriptif qui apparaît dans l'**Explorateur Windows**.

HelpFile, propriété

La propriété **HelpFile** dans Visual Basic 6.0 spécifie un fichier d'aide au niveau de l'application.

Dans Visual Basic 2005, les fichiers d'aide sont spécifiés sur une base par-formulaire à l'aide d'un composant HelpProvider. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.

Version, propriétés

Les propriétés **Major**, **Minor** et **Revision** dans Visual Basic 6.0 sont utilisés pour retourner un numéro de version à trois parties pour une application.

Dans Visual Basic 2005, les informations sur la version incluent également une propriété Build et le numéro de version à quatre parties complet peut être récupéré à l'aide d'une seule propriété **Version**. Pour plus d'informations, consultez Numérotation des versions pour les utilisateurs de Visual Basic 6.0.

Title, propriété

La propriété **Title** de l'objet **App** dans Visual Basic 6.0 peut être modifiée par programme ; cela permet de déterminer le nom qui apparaît dans le **Gestionnaire des tâches** de Windows.

Dans Visual Basic 2005, l'attribut **Title** est en lecture seule. La propriété **Text** du formulaire principal de l'application détermine le nom qui est affiché dans le **Gestionnaire des tâches**. Vous pouvez définir la propriété **Text** par programme, mais le texte dans la barre de titre du formulaire est également modifié.

TaskVisible, propriété

Dans Visual Basic 6.0, la propriété **TaskVisible** de l'objet **App** détermine si une application apparaît dans la liste des tâches de Windows (Windows 9*x*) ou dans l'onglet **Applications du Gestionnaire des Tâches** (Windows 2000 ou versions supérieures). La propriété est couramment utilisée pour empêcher un utilisateur de fermer une application qui a été conçue pour s'exécuter comme tâche en arrière-plan. Dans la plupart des cas, la propriété **TaskVisible** est utilisée avec les applications qui n'affichent pas d'interfaces utilisateur.

Dans Visual Basic 2005, il n'existe pas d'équivalent de la propriété **TaskVisible** ; vous pouvez néanmoins créer un service Windows ou une application console qui n'apparaîtra pas dans la liste des tâches.

PrevInstance, propriété

La propriété **PrevInstance** dans Visual Basic 6.0 détermine si une instance de l'application est déjà en cours d'exécution. En

général, vous devez appeler **PrevInstance** au démarrage de l'application, puis abandonner l'application si la valeur **true** est retournée.

Il n'y a aucun équivalent direct pour **PrevInstance** dans Visual Basic 2005. Vous devez contrôler si plusieurs instances d'une application sont autorisées ou pas en définissant la propriété **Plusieurs instances** sur l'onglet **Application** du **Concepteur de projets**. Au moment de l'exécution, un événement **NextInstanceStarted** est déclenché lorsqu'un utilisateur essaie de démarrer une autre instance de l'application. Vous pouvez ajouter le code à ce gestionnaire d'événements pour activer la première instance de l'application.

Différences supplémentaires

Un certain nombre de propriétés de l'objet **App** dans Visual Basic 6.0 se rapporte aux composants ActiveX, à OLE automation ainsi qu'à d'autres technologies qui, aujourd'hui, sont obsolètes et ne sont pas prises en charge dans Visual Basic 2005. Elles sont indiquées dans le tableau Équivalences des propriétés de l'objet App plus tard dans cette rubrique.

Modifications du code de l'objet App

Les exemples de code suivants illustrent les différences entre les techniques de codage dans Visual Basic 6.0 et dans Visual Basic 2005 pour certaines utilisations courantes de l'objet **App**.

Modifications du code pour afficher le numéro de version d'une application

L'exemple suivant illustre la récupération du numéro de version d'une application et l'affichage de celui-ci dans une étiquette.

```
' Visual Basic 6.0
Label1.Caption = "Version: " & App.Major & "." & App.Minor & "." _
& App.Revision
```

VB

```
' Visual Basic 2005
Label1.Text = My.Application.Info.Version.ToString()
```

Modifications du code pour déterminer le chemin d'accès d'une application

L'exemple suivant illustre comment afficher un fichier image qui est stocké dans le dossier de l'application en récupérant le chemin d'accès vers l'application en cours d'exécution.

```
' Visual Basic 6.0
Picture1.Picture = LoadPicture(App.Path & "\Logo.jpg")
```

VB

```
' Visual Basic 2005
PictureBox1.Image = System.Drawing.Bitmap.FromFile( _
  My.Application.Info.DirectoryPath & "\Logo.jpg")
```

Modifications du code pour empêcher une deuxième instance d'une application de s'exécuter

L'exemple suivant illustre l'implémentation d'une application à instance unique.

```
' Visual Basic 6.0
Private Sub Form_Load()
    If App.PrevInstance = True Then
        MsgBox("The application is already running!")
    End If
End Sub
```

VB

```
' Visual Basic 2005
' Assumes that the Make Single Instance Application checkbox in the
' Project Designer has been checked.
```

```
Private Sub MyApplication_StartupNextInstance(ByVal sender As Object, _
ByVal e As Microsoft.VisualBasic.ApplicationServices. _
StartupNextInstanceEventArgs) Handles Me.StartupNextInstance
    MsgBox("The application is already running!")
End Sub
```

Équivalents des propriétés de l'objet App

Le tableau suivant répertorie toutes les propriétés et méthodes de l'objet **App** dans Visual Basic 6.0 ainsi que leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
Comments	My.Application.Info.Description, propriété
CompanyName	My.Application.Info.CompanyName, propriété
EXEName	My.Application.Info.AssemblyName, propriété
FileDescription	My.Application.Info.Title, propriété
HelpFile	Nouvelle implémentation. Utilisez un composant **HelpProvider**. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HInstance	GetHINSTANCE
LegalCopyright	My.Application.Info.Copyright, propriété
LegalTrademarks	My.Application.Info.Trademark, propriété
LogEvent method	WriteEntry, méthode (My.Application.Log et My.Log)
LogMode **LogPath**	Nouvelle implémentation. Pour plus d'informations, consultez Enregistrement d'informations provenant de l'application.
Major	My.Application.Info.Version, propriété **Remarque** Le format pour les numéros de version est différent dans Visual Basic 2005. Pour plus d'informations, consultez Numérotation des versions pour les utilisateurs de Visual Basic 6.0.
Minor	My.Application.Info.Version, propriété **Remarque** Le format pour les numéros de version est différent dans Visual Basic 2005. Pour plus d'informations, consultez Numérotation des versions pour les utilisateurs de Visual Basic 6.0.
NonModalAllowed	Nouvelle implémentation. Il s'agit d'une propriété en lecture seule associée aux fichiers .dll ActiveX. Le Common Language Runtime gère automatiquement ce comportement.

OleRequestPendingMsgText **OleRequestPendingMsgTitle** **OleRequestPendingTimeout** **OleServerBusyMsgText** **OleServerBusyMsgTitle** **OleServerBusyRaiseError** **OleServerBusyTimeout**	Pas d'équivalent. Ces propriétés se rapportent à OLE automation qui n'est pas pris en charge par Visual Basic 2005.
Path	My.Application.Info.DirectoryPath, propriété
PrevInstance	IsSingleInstance **Remarque** La propriété **IsSingleInstance** est une propriété **Protected**. Elle ne peut être définie qu'au moment du design dans le **Concepteur de projets** en activant ou désactivant la case à cocher **Plusieurs instances**.
ProductName	My.Application.Info.ProductName, propriété
RetainedProject	Nouvelle implémentation. Visual Basic 2005 ne peut pas conserver un projet en mémoire.
Revision	My.Application.Info.Version, propriété **Remarque** Le format pour les numéros de version est différent dans Visual Basic 2005. Pour plus d'informations, consultez Numérotation des versions pour les utilisateurs de Visual Basic 6.0.
StartLogging method	Nouvelle implémentation. Pour plus d'informations, consultez Enregistrement d'informations provenant de l'application.
StartMode	Pas d'équivalent. Cette propriété concerne la création de composants ActiveX qui n'est pas prise en charge dans Visual Basic 2005.
TaskVisible	Nouvelle implémentation. Pour créer une application qui n'apparaît pas dans le **Gestionnaire des tâches**, créez un projet **Service Windows** ou **Application console** .
ThreadID	Nouvelle implémentation. Cette propriété a retourné l'ID du thread en cours d'exécution; le modèle de thread est considérablement différent dans Visual Basic 2005.
Title	My.Application.Info.Title, propriété
UnattendedApp	Nouvelle implémentation. Pour les applications sans assistance dans Visual Basic 2005, créez un projet **Application console**.

Mettre à niveau des remarques

Lorsqu'une application est mise à niveau depuis Visual Basic 6.0, seules les propriétés **Major** et **Minor** sont mises à niveau ; Visual Basic 2005 assigne de nouvelles valeurs aux propriétés Revision et **Build**.

Voir aussi

Référence

My.Application, objet

Concepts

Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0

Numérotation des versions pour les utilisateurs de Visual Basic 6.0

Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0

Cette rubrique compare le contrôle **CheckBox** de Visual Basic 6.0 avec son équivalent dans Visual Basic 2005.

Le contrôle **CheckBox** de Visual Basic 6.0 est remplacé par le contrôle CheckBox Windows Forms dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, leur comportement est également différent.

Différences conceptuelles

Click, événement

Dans Visual Basic 6.0, l'événement **Click** est déclenché lorsque l'état **CheckBox** est modifié par programme. Dans Visual Studio 2005, vous utilisez l'événement CheckStateChanged et non l'événement **Click**.

Différences supplémentaires

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris les différences relatives à la liaison de données, la gestion des polices, le glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Vue d'ensemble des Windows Forms et Nouveautés dans les Windows Forms et les contrôles Windows Forms.

Modifications apportées au code du contrôle CheckBox

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications apportées au code de réponse à un clic du contrôle CheckBox

Le code suivant décrit la modification du texte d'un contrôle **CheckBox** lorsqu'un utilisateur active ou désactive le contrôle. La propriété de Visual Basic 6.0 **Value** est remplacée par la propriété Checked de Visual Basic 2005 et la propriété **Caption** est remplacée par la propriété Text. Remarquez que l'exemple de Visual Basic 2005 utilise l'événement **CheckStateChanged** pour dupliquer le comportement de l'événement **Click** de Visual Basic 6.0.

```
' Visual Basic 6.0
Private Sub Check1_Click()
    If Check1.Value = 1 Then
        Check1.Caption = "Checked"
    Else
        Check1.Caption = "Unchecked"
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub CheckBox1_CheckStateChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles CheckBox1.CheckStateChanged
    If CheckBox1.Checked = True Then
        CheckBox1.Text = "Checked"
    Else
        CheckBox1.Text = "Unchecked"
    End If
End Sub
```

Modifications apportées au code de détermination de l'état d'une case à cocher

L'exemple suivant décrit la vérification de l'état d'un contrôle **CheckBox** au moment de l'exécution.

```
' Visual Basic 6.0
Select Case Check1.Value
    Case vbUnchecked
        Check1.Caption = "Unchecked"
    Case vbChecked
        Check1.Caption = "Checked"
    Case vbGrayed
        Check1.Caption = "Disabled"
```

```
End Select
```

VB

```
' Visual Basic 2005
Select Case CheckBox1.CheckState
    Case CheckState.Unchecked
        CheckBox1.Text = "Unchecked"
    Case CheckState.Checked
        CheckBox1.Text = "Checked"
    Case CheckState.Indeterminate
        CheckBox1.Text = "Disabled"
End Select
```

Propriété, méthode et événement équivalents du contrôle CheckBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes noms et comportements ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Propriété de Visual Basic 6.0	**Équivalent Visual Basic 2005**
Alignment **0** (Justifier à gauche) **1** (Justifier à droite)	CheckAlign Valeur d'énumération MiddleLeft Valeur d'énumération MiddleRight
Appearance **0** (À deux dimensions) **1** (3D)	Énumération FlatStyle Valeur d'énumération Flat Valeur d'énumération Standard
BackColor	BackColor **Remarque** Pour obtenir la liste des constantes de couleur, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0. **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Caption	**Text**
Container	Parent

DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DisabledPicture **DownPicture**	Nouvelle implémentation. Pour plus d'informations, consultez Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Pour obtenir la liste des constantes de couleur, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0. **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height , Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.

Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
MaskColor	Nouvelle implémentation. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
Picture	Image
RightToLeft **True**	RightToLeft Valeur d'énumération Yes Valeur d'énumération No
Style **0** (Standard) **1** (Graphique)	Appearance Valeur d'énumération Normal Valeur d'énumération Button **Remarque** Il existe des différences relatives à la gestion du style **Graphical** dans Visual Basic 2005. Pour plus d'informations, consultez Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau.
Tag	Tag
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
UseMaskColor	Nouvelle implémentation. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0.

Value **0** (**vbUnchecked**) **1** (**vbChecked**) **2** (**vbGrayed**)	CheckState Valeur d'énumération Unchecked Valeur d'énumération Checked Valeur d'énumération Indeterminate
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width **, Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Méthode de Visual Basic 6.0	Équivalent Visual Basic 2005
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder **0** (**vbBringToFront**) **1** (**vbSendToBack**)	BringToFront, SendToBack

Événements

Événement de Visual Basic 6.0	Équivalent Visual Basic 2005
Click	**CheckStateChanged** **Remarque** Dans Visual Basic 6.0, l'événement **Click** est déclenché lorsque l'état **CheckBox** est modifié par programme. Dans Visual Basic 2005, l'événement **Click** n'est pas déclenché ; par conséquent, vous devez utiliser l'événement **CheckStateChanged**.

DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 à l'aide de l'Assistant Mise à niveau, les contrôles **CheckBox** sont mis à niveau vers des contrôles **CheckBox** Windows Forms et le code est mis à jour pour utiliser des propriétés, des méthodes et des événements équivalents. Aux emplacements dans lesquels il n'y a aucun équivalent ou il existe des différences de comportement potentielles, les commentaires sont insérés dans le code avec un lien vers une rubrique d'aide.

Voir aussi

Référence

Vue d'ensemble du contrôle CheckBox (Windows Forms)

Concepts

Accès aux données pour les utilisateurs de Visual Basic 6.0
Propriété Style pour les utilisateurs de Visual Basic 6.0
Gestion des polices pour les utilisateurs de Visual Basic 6.0
Système de coordonnées pour les utilisateurs de Visual Basic 6.0
Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0
Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0
MousePointer pour les utilisateurs de Visual Basic 6.0
Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Objet Clipboard pour les utilisateurs de Visual Basic 6.0

L'objet **Clipboard** dans Visual Basic 6.0 est remplacé par un objet **Clipboard** similaire dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, l'objet **Clipboard** est utilisé pour stocker et récupérer du texte, des images et des données du Presse-papiers système.

Visual Basic 2005 a un objet **Clipboard** situé dans l'espace de noms **My.Computer** doté de quelques nouvelles méthodes et de quelques méthodes légèrement différentes.

Formats de données du Presse-papiers

Dans Visual Basic 6.0, les constantes permettent de définir le type de données passées à ou récupérées du **Clipboard**.

Visual Basic 2005 utilise un objet DataFormats pour définir le type de données ; plusieurs nouveaux formats de données sont pris en charge. Une liste de constantes de format de Presse-papiers Visual Basic 6.0 et leurs équivalents Visual Basic 2005 est fournie plus tard dans cette rubrique.

Modifications apportées au code de l'objet du Presse-papiers

L'exemple suivant montre comment stocker et récupérer le texte à l'aide du **Clipboard**.

```
' Visual Basic 6.0
Clipboard.Clear
Clipboard.SetText "Hello", vbCFText
If Clipboard.GetFormat(vbCFText) Then
   Text1.Text = Clipboard.GetText(vbCFText)
End If
```

VB

```
' Visual Basic 2005
My.Computer.Clipboard.Clear()
My.Computer.Clipboard.SetText("Hello")
If My.Computer.Clipboard.ContainsText Then
  TextBox1.Text = My.Computer.Clipboard.GetText
End If
```

Équivalences des constantes Clipboard

Visual Basic 6.0	Équivalent Visual Basic 2005
vbCFBitmap	Bitmap
vbCFDIB	Dib
vbCFEMetafile	EnhancedMetafile
vbCFFiles	FileDrop
vbCFLink	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
vbCFMetafile	MetafilePict
vbCFPalette	Palette

vbCFRTF	Rtf
vbCFText	Text

Remarque

L'objet **DataFormats** fait partie de l'espace de noms System.Windows.Forms. En plus des formats répertoriés ci-dessus, **DataFormats** prend en charge plusieurs formats de Presse-papiers qui ne l'étaient pas dans Visual Basic 6.0.

Voir aussi

Tâches

Comment : effacer le Presse-papiers en Visual Basic
Comment : lire le Presse-papiers en Visual Basic
Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic
Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic
Comment : récupérer une image du Presse-papiers en Visual Basic
Comment : écrire dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard, membres de l'objet

Contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **ComboBox** dans Visual Basic 6.0 est remplacé par le contrôle ComboBox Windows Forms dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Change, événement

Dans Visual Basic 6.0, l'événement **Change** du contrôle **ComboBox** est déclenché lorsque le texte du contrôle est modifié. L'événement **Change** n'est pas déclenché lorsqu'un élément est sélectionné dans la partie liste du contrôle. La modification par programme du texte d'un élément dans la liste ne déclenche pas l'événement **Change**.

Dans Visual Basic 2005, l'événement **Change** n'existe plus ; l'événement TextChanged est déclenché chaque fois que le texte est modifié pour une raison ou pour une autre, telle que :

- Lorsque la partie de texte est modifiée
- Lorsqu'un élément est sélectionné dans la liste
- Lorsqu'un élément de liste est modifié par programme
- Lorsque la méthode **AddItem** est appelée

ItemData, propriété

Dans Visual Basic 6.0, la propriété **ItemData** d'un contrôle **ComboBox** peut être définie au moment du design dans la fenêtre **Propriétés** pour associer un **Integer** à un élément **ComboBox**.

Dans Visual Basic 2005, la propriété **ItemData** n'existe plus. La bibliothèque de compatibilité Visual Basic (Microsoft.VisualBasic.Compatibility) contient une fonction **SetItemData** qui peut être utilisée pour associer **Integer** à un élément ; la fonction **GetItemData** peut être utilisée pour récupérer l'élément.

Locked, propriété

Dans Visual Basic 6.0, la propriété **Locked** d'un contrôle **ComboBox** détermine si la zone de texte du contrôle peut être modifiée.

Dans Visual Basic 2005, la propriété **Locked** empêche un contrôle d'être déplacé au moment du design. Il n'y a aucun équivalent direct pour la propriété **Locked** dans Visual Basic 6.0 ; toutefois, vous pouvez obtenir le même effet en affectant DropDownList à la propriété DropDownStyle du contrôle **ComboBox**.

Remarque Dans Visual Basic 6.0, l'affectation de la valeur **True** à la propriété **Locked** empêche également que la sélection ne soit modifiée. Vous pouvez dupliquer ce comportement en annulant la sélection opérée dans l'événement `MouseDown`.

NewIndex, propriété

Dans Visual Basic 6.0, la propriété **NewIndex** était utilisée pour récupérer l'index du dernier élément ajouté à un contrôle **ComboBox**.

Dans Visual Basic 2005, la propriété NewIndex n'existe plus. Vous pouvez utiliser la valeur de retour de la méthode **Item.Add** pour récupérer l'index d'un élément tel qu'il est ajouté.

TopIndex, propriété

Dans Visual Basic 6.0, la propriété **TopIndex** est utilisée pour retourner ou définir une valeur désignant l'élément d'un contrôle **ComboBox** ou **ListBox** qui est affiché en première position de la liste. Cette propriété est couramment utilisée pour faire défiler une liste sans y sélectionner d'élément.

Dans Visual Basic 2005, la propriété **ComboBox** ne prend plus en charge la propriété **TopIndex**. Dans la plupart des cas, cela ne devrait poser aucun problème, dans la mesure où la définition de la propriété **TopIndex** n'a d'effet visible que si la propriété **Style** a la valeur **1 - SimpleCombo**. Dans le cas présent, vous pouvez émuler ce comportement à l'aide d'un contrôle ListBox et d'un contrôle TextBox ; le contrôle **ListBox** prend toujours en charge la propriété TopIndex.

Scroll, événement

L'événement **Scroll** dans Visual Basic 6.0 est utilisé conjointement avec la propriété **TopIndex** pour exécuter des actions lors du défilement de la liste. Visual Basic 2005 ne prend pas en charge l'événement **Scroll** ; toutefois, dans la plupart des cas, l'événement SelectedIndexChanged est un élément de remplacement approprié.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, les opérations de glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications du code pour le contrôle ComboBox

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour mettre un contrôle ComboBox en lecture seule

Le code suivant illustre comment empêcher un utilisateur d'ajouter de nouveaux éléments à un contrôle **ComboBox**.

```
' Visual Basic 6.0
Combo1.Locked = True
```

VB

```
' Visual Basic 2005
ComboBox1.DropDownStyle = ComboBoxStyle.DropDownList
```

Modifications du code pour sélectionner l'élément ajouté en dernier dans un contrôle ComboBox

Le code suivant illustre comment définir la sélection à un élément qui a été ajouté par programme à un contrôle **ComboBox**.

```
' Visual Basic 6.0
Combo1.AddItem "This is a new item"
Combo1.ListIndex = Combo1.NewIndex
```

VB

```
' Visual Basic 2005
Dim i As Integer
i = ComboBox1.Items.Add("This is a new item")
ComboBox1.SelectedIndex = i
```

Modifications du sous code privé pour associer ItemData

Le code suivant illustre comment associer un numéro d'employé à chaque employé répertorié dans un contrôle **ComboBox**, puis comment récupérer ce numéro au moment de l'exécution.

```
' Visual Basic 6.0
Private Sub Form_Load
    Combo1.AddItem "Nancy Davolio"
    Combo1.ItemData(Combo1.NewIndex) = 12345
    Combo1.AddItem "Judy Phelps"
    Combo1.ItemData(Combo1.NewIndex) = 67890
End Sub
Private Sub Combo1_Click()
    Label1.Caption = "Empoyee #" & _
    CStr(Combo1.ItemData(Combo1.ListIndex))
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load

   ComboBox1.Items.Add("Nancy Davolio")
   Microsoft.VisualBasic.Compatibility.VB6. _
      SetItemData(ComboBox1, ComboBox1.Items.Count() - 1, 12345)
```

```
    ComboBox1.Items.Add("Judy Phelps")
    Microsoft.VisualBasic.Compatibility.VB6. _
       SetItemData(ComboBox1, ComboBox1.Items.Count() - 1, 67890)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged

    Label1.Text = "Employee #" & CStr( _
    Microsoft.VisualBasic.Compatibility.VB6. _
       GetItemData(ComboBox1, ComboBox1.SelectedIndex))
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle ComboBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Contrôle de données pour les utilisateurs de Visual Basic 6.0.

DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
ItemData	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de mettre à niveau la propriété ItemData.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
List	Items

ListCount	Count
ListIndex	SelectedIndex
Locked	**DropDownStyle** = **DropDownList** **Remarque** Dans Visual Basic 6.0, l'affectation de la valeur **True** à la propriété **Locked** empêche également que la sélection ne soit modifiée. Dans Visual Basic 2005, vous pouvez dupliquer ce comportement en annulant la sélection opérée dans l'événement MouseDown.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
NewIndex	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de mettre à niveau la propriété NewIndex.
OLEDragMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
RightToLeft	RightToLeft
SelLength	SelectionLength
SelStart	SelectionStart
SelText	SelectedText
Style	**DropDownStyle**
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
TopIndex	Nouvelle implémentation. Pour plus d'informations, consultez Propriété TopIndex et événement Scroll pour les utilisateurs de Visual Basic 6.0.

WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	Équivalent Visual Basic 2005
AddItem	Add AddRange Insert
Clear	Clear
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
RemoveItem	**Items.Remove**
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	BringToFront Fonctions SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005
Change	**TextChanged** **Remarque** Le comportement de l'événement **TextChanged** est légèrement différent. Pour plus d'informations, consultez Événement Change du contrôle ComboBox pour les utilisateurs de Visual Basic 6.0.
Click	**SelectedIndexChanged**

DblClick	Nouvelle implémentation. Utilisez l'événement **SelectedIndexChanged** ou **TextChanged**.
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Scroll	Nouvelle implémentation. Pour plus d'informations, consultez Propriété TopIndex et événement Scroll pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, l'événement **Change** du contrôle **ComboBox** est mappé à l'événement **TextChanged** du contrôle Visual Basic 2005 **ComboBox**. Le comportement de l'événement **TextChanged** diffère de celui de l'événement **Change** ; cette différence peut produire des résultats imprévus dans votre code.

Tout code qui fait référence aux propriétés **ItemData**, **Locked**, **NewIndex** ou **TopIndex** ou tout code dans la procédure événementielle **Scroll** n'est pas mis à niveau ; des commentaires d'avertissement sont ajoutés au code et ce dernier doit être supprimé ou modifié avant la compilation de l'application.

Voir aussi

Référence

Vue d'ensemble du contrôle ComboBox (Windows Forms)

Concepts

Comportement des couleurs pour les utilisateurs de Visual Basic 6.0
Objet Font pour les utilisateurs de Visual Basic 6.0
MousePointer pour les utilisateurs de Visual Basic 6.0
Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0

Contrôle CommandButton pour les utilisateurs de Visual Basic 6.0

Le contrôle **CommandButton** dans Visual Basic 6.0 est remplacé par le contrôle Button Windows Forms dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Default et Cancel, propriétés

Dans Visual Basic 6.0, les propriétés booléennes **Default** et **Cancel** d'un contrôle **CommandButton** détermine si le contrôle répond respectivement aux touches ENTRÉE ou ÉCHAP.

Dans Visual Basic 2005, le contrôle **Button** ne dispose plus des propriétés **Default** ou **Cancel**. L'objet Form dispose des propriétés AcceptButton et CancelButton qui fournissent les mêmes fonctionnalités ; ces propriétés prennent le nom d'un contrôle **Button** comme argument.

Value, propriété

Dans Visual Basic 6.0, la propriété booléenne **Value** d'un contrôle **CommandButton** indique si le contrôle a été sélectionné ; l'affectation de **True** à la propriété appelle l'événement **Click** du bouton.

Dans Visual Basic 2005, le contrôle **Button** ne dispose plus d'une propriété **Value**. Vous pouvez utiliser l'événement GotFocus pour déterminer si le contrôle est sélectionné ; la méthode PerformClick peut être utilisée pour appeler l'événement Click.

Style, Picture, DownPicture et DisabledPicture, propriétés

Dans Visual Basic 6.0, l'affectation de la valeur **1 – Graphical** à la propriété **Style** d'un contrôle **CommandButton** modifie l'apparence de ce contrôle pour permettre l'affichage d'images. Les propriétés **Picture**, **DownPicture** et **DisabledPicture** sont utilisées pour assigner des images à afficher en réponse à des changements d'état. Par exemple, lorsque l'utilisateur clique sur un contrôle **CommandButton**, l'image **DownPicture** est affichée ; si le contrôle est désactivé, l'image **DisabledPicture** est affichée.

Visual Basic 2005 ne prend plus en charge les propriétés **Style**, **Picture**, **DownPicture** ou **DisabledPicture**. La propriété **Style** n'est plus nécessaire ; l'assignation d'une image à la propriété Image a le même effet que l'affectation de **Graphical** à la propriété **Style**. La propriété **Picture** est remplacée par la propriété **Image**. Les fonctionnalités des propriétés **DownPicture** et **DisabledPicture** peuvent être émulées à l'aide d'un contrôle ImageList qui contient plusieurs images.

MaskColor, propriété

Dans Visual Basic 6.0, la propriété **MaskColor** d'un contrôle **CommandButton** est utilisée pour définir une couleur qui devient transparente, permettant ainsi l'affichage d'une image d'arrière plan. Pour utiliser cette propriété, **Graphical** doit être affecté à la propriété **Style**, **True** à la propriété **UseMaskColor**, et une bitmap doit être assignée à la propriété **Picture**.

Dans Visual Basic 2005, il n'y a aucun équivalent direct à la propriété **MaskColor**. Cependant, vous pouvez définir la transparence d'un contrôle à l'aide de la méthode MakeTransparent d'un objet Bitmap.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris les différences relatives à la liaison de données, la gestion des polices, le glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications apportées au code du contrôle CommandButton

Les exemples de code suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications apportées au code pour définir les boutons Par défaut et Annuler

L'exemple suivant montre comment définir deux boutons pour qu'ils soient les boutons Par défaut et Annuler d'un formulaire.

```
' Visual Basic 6.0
' Set the first button to respond to the Enter key.
Command1.Default = True
' Set the second button to respond to the Esc key.
Command2.Cancel = True
```

VB

```
' Visual Basic 2005
' Set the first button to respond to the Enter key.
Me.AcceptButton = Button1
' Set the second button to respond to the Esc key.
Me.CancelButton = Button2
```

Modifications apportées au code pour ajouter la transparence à un bouton

L'exemple suivant illustre l'utilisation de méthodes permettant de définir une région transparente sur un bouton qui contient une image ; toute partie de l'image qui est blanche deviendra transparente.

```
' Visual Basic 6.0
' Assumes a picture has been assigned to the Picture property
' and that the Style property has been set to Graphical.
Command1.UseMaskColor = True
Command1.MaskColor = vbWhite
```

VB

```
' Visual Basic 2005
' Assumes a picture has been assigned to the BackgroundImage property.
Dim ButtonBitmap As New System.Drawing.Bitmap(Button1.BackgroundImage)
ButtonBitmap.MakeTransparent(System.Drawing.Color.White)
Button1.BackgroundImage = ButtonBitmap
```

Équivalences des propriétés, méthodes et événements du contrôle CommandButton

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes noms et comportements ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ces tableaux répertorient les liens vers les rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Cancel	**CancelButton** (objet **Form**)
Caption	Text
Container	Parent
Default	**AcceptButton** (objet **Form**)

DisabledPicture **DownPicture**	Nouvelle implémentation. Pour plus d'informations, consultez Propriété Style pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
MaskColor	Nouvelle implémentation. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0 .
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Parent	Méthode FindForm
Picture	**Image**
RightToLeft	RightToLeft
Style	Nouvelle implémentation. Pour plus d'informations, consultez Propriété Style pour les utilisateurs de Visual Basic 6.0.
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
UseMaskColor	Nouvelle implémentation. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0.
Value	Nouvelle implémentation. La méthode **PerformClick** est identique à l'affectation de **True** à **Value** ; il n'y a aucun équivalent pour les autres utilisations de la propriété **Value**.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.

ZOrder	Fonction BringToFront ou SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 à l'aide de l'Assistant Mise à niveau, les contrôles **CommandButton** sont mis à niveau vers des contrôles **Button** Windows Forms et le code est mis à jour pour utiliser les propriétés, méthodes et événements équivalents. Lorsqu'il n'y a pas d'équivalents ou qu'il existe des différences de comportement potentielles, des commentaires sont insérés dans le code avec des liens vers les rubriques d'aide.

Lorsqu'une application est mise à niveau vers Visual Basic 2005, si la propriété **Style** a la valeur **1 -Graphical**, la propriété **FlatStyle** du contrôle mis à niveau a la valeur **Standard** et toute image assignée à la propriété **Picture** au moment du design est assignée à la propriété `Image` du contrôle mis à niveau.

Si les propriétés **DownPicture** ou **DisabledPicture** ont été définies au moment du design ou de l'exécution, vous devrez modifier votre application mise à niveau pour utiliser un contrôle **ImageList**. Pour plus d'informations, consultez Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau.

L'Assistant Mise à niveau ne met pas à niveau le code qui utilise la propriété **MaskColor**. Des avertissements de mise à niveau seront insérés dans votre code. Vous devrez modifier le code avant d'exécuter l'application.

Voir aussi

Concepts

MaskColor pour les utilisateurs de Visual Basic 6.0

Contrôle CommonDialog pour les utilisateurs de Visual Basic 6.0

Le contrôle **CommonDialog** de Visual Basic 6.0 est remplacé par plusieurs composants spécialisés dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, le contrôle ActiveX **CommonDialog** est utilisé pour afficher plusieurs boîtes de dialogue courantes (**Ouvrir**, **Enregistrer**, **Couleur**, **Police**, **Imprimer** et **Aide**) dans votre application.

Dans Visual Basic 2005, le contrôle **CommonDialog** est remplacé par des composants pour afficher des boîtes de dialogue : les composants OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog et PrintDialog.

Remarque

Il n'existe pas d'équivalent direct permettant d'afficher une boîte de dialogue de l'aide dans Visual Basic 2005. Le contrôle **CommonDialog** ne prenait en charge que l'aide Windows ; Visual Basic 2005 ne prend en charge que l'aide HTML. Visual Basic 2005 utilise le composant HelpProvider pour afficher l'aide pour votre application. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.

Modifications du code pour le contrôle CommonDialog

Les exemples suivants illustrent les différences entre les techniques de codage dans Visual Basic 6.0 et dans Visual Basic 2005 pour certaines utilisations courantes de l'objet **CommonDialog**.

Modifications du code pour afficher une boîte de dialogue Ouvrir

Le code suivant illustre l'affichage d'une boîte de dialogue **Ouvrir**, initialisée dans le répertoire Program Files.

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.InitDir = "C:\Program Files"
CommonDialog1.ShowOpen
```

VB

```
' Visual Basic 2005
' Uses a OpenFileDialog component.
OpenFileDialog1.InitialDirectory = "C:\Program Files"
OpenFileDialog1.ShowDialog()
```

Modifications du code pour afficher une boîte de dialogue Enregistrer

Le code suivant illustre l'affichage d'une boîte de dialogue **Enregistrer** en enregistrant le fichier dans le dossier de l'application.

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.InitDir = App.Path
CommonDialog1.ShowSave
```

VB

```
' Visual Basic 2005
' Uses a SaveFileDialog component.
SaveFileDialog1.InitialDirectory = My.Application.Info.DirectoryPath
SaveFileDialog1.ShowDialog()
```

Modifications du code pour afficher une boîte de dialogue Imprimer

Le code suivant illustre l'affichage d'une boîte de dialogue **Imprimer**, en imprimant un fichier localisé dans le dossier de l'application.

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.FileName = App.Path & "MyFile.txt"
CommonDialog1.ShowPrinter
```

VB

```
' Visual Basic 2005
' Uses PrintDocument and PrintDialog components.
PrintDocument1.DocumentName = My.Application.Info.DirectoryPath _
& "MyFile.txt"
PrintDialog1.Document = PrintDocument1
PrintDialog1.ShowDialog()
```

Modifications du code pour afficher l'aide

Le code suivant illustre l'affichage d'un fichier d'aide de votre application, en l'ouvrant dans la table des matières.

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.HelpFile = "C:\Windows\Help\calc.hlp"
CommonDialog1.HelpCommand = cdlHelpContents
CommonDialog1.ShowHelp
```

VB

```
' Visual Basic 2005
' Uses the Help.ShowHelp method.
Help.ShowHelp(Me, "file://C:\Windows\Help\calc.chm", _
HelpNavigator.TableOfContents)
```

Équivalences des méthodes et des propriétés du contrôle CommonDialog

Les tableaux suivants répertorient les propriétés et les méthodes Visual Basic 6.0 ainsi que leurs équivalents dans Visual Basic 2005. Ces propriétés et méthodes qui ont les mêmes noms et comportements ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Action	Nouvelle implémentation. La propriété **Action** de Visual Basic 6.0 détermine la boîte de dialogue à afficher ; Visual Basic 2005 utilise un composant séparé pour chaque boîte de dialogue.
CancelError	Cancel
Copies	Copies
DialogTitle	Title (composants **OpenFileDialog** et **SaveFileDialog** uniquement) Nouvelle implémentation pour les autres composants. Les titres des fenêtres standard (**Color**, **Font** et **Print**) sont affichés et ne peuvent pas être substitués.

FileName	FileNames
FileTitle	Nouvelle implémentation. La propriété **FileTitle** de Visual Basic 6.0 retourne le **FileName** sans le chemin d'accès ; vous pouvez analyser la propriété **FileNames** pour obtenir le nom sans le chemin d'accès.
Flags	La propriété **Flags** de Visual Basic 6.0 fournit des constantes pour la définition de plusieurs attributs de différentes boîtes de dialogue courantes. Plutôt que d'utiliser des constantes, les composants de dialogue fournissent des propriétés pour définir les attributs.
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
FromPage	FromPage
hDC	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
HelpCommand	HelpNavigator
HelpFile	HelpNamespace
HelpKey	Paramètres *parameter* de la méthode ShowHelp.
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
InitDir	InitialDirectory
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Max	MaxSize (composant **FontDialog**) MaximumPage (composant **PrintDialog**)
Min	MinSize (composant **FontDialog**) MinimumPage (composant **PrintDialog**)
MaxFileSize	Nouvelle implémentation. Cette propriété Visual Basic 6.0 alloue la mémoire pour les noms de fichiers extrêmement longs ; elle n'est donc plus nécessaire dans le code managé.
Orientation	Landscape
Parent	Méthode FindForm
PrinterDefault	Nouvelle implémentation. Cette propriété Visual Basic 6.0 est utilisée conjointement avec la propriété **hDC** pour imprimer à l'aide de méthodes d'interface de périphérique graphique ; cette propriété n'est plus prise en charge.
Top	**P:System.Windows.Forms.Control.Top** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
ToPage	ToPage

Méthodes

Visual Basic 6.0	Équivalent Visual Basic 2005
AboutBox	Nouvelle implémentation. La propriété **AboutBox** affichait une boîte **À propos de** pour le contrôle **CommonDialog** qui était créé pour Microsoft par un tiers.
ShowColor	ShowDialog (composant **ColorDialog**)
ShowFont	**ShowDialog** (composant **FontDialog**)
ShowHelp	ShowHelp
ShowOpen	**ShowDialog** (composant **OpenFileDialog**)
ShowPrinter	**ShowDialog** (composant **PrintDialog**)
ShowSave	**ShowDialog** (composant **SaveFileDialog**)

Mettre à niveau des remarques

Lors de la mise à niveau d'une application de Visual Basic 6.0 vers Visual Basic 2005, tous les contrôles **CommonDialog** sont mis à niveau vers leur composant de dialogue équivalent (**OpenFileDialog**, **SaveFileDialog**, **ColorDialog**, **FontDialog** ou **PrintDialog**).

Il n'existe pas d'équivalent direct permettant d'afficher une boîte de dialogue **Help** dans Visual Basic 2005. Si un contrôle **CommonDialog** est utilisé comme boîte de dialogue **Help**, un avertissement est émis et il n'est pas mis à niveau. Pour plus

d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.

Un seul contrôle **CommonDialog** pouvait être utilisé pour afficher différents types de boîtes de dialogue. Dans ce cas, il est mis à niveau vers un composant **OpenFileDialog** ; un avertissement est émis et vous devez ajouter les autres composants de dialogue manuellement.

De plus, lorsqu'un contrôle **CommonDialog** était utilisé comme variable, il se peut que vous deviez le modifier pour faire référence à un composant de dialogue spécifique après la mise à niveau.

Voir aussi

Référence

Vue d'ensemble du composant FontDialog (Windows Forms)
Vue d'ensemble du composant HelpProvider (Windows Forms)

Autres ressources

ColorDialog, composant (Windows Forms)
OpenFileDialog, composant (Windows Forms)
SaveFileDialog, composant (Windows Forms)
PrintDialog, composant (Windows Forms)
PrintDocument, composant (Windows Forms)

Collection Controls pour les utilisateurs de Visual Basic 6.0

La collection **Controls** de Visual Basic 6.0 est remplacée par la classe ControlCollection dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, la collection **Controls** est une collection avec des éléments qui représentent les contrôles d'un formulaire ou d'un contrôle conteneur.

Dans Visual Basic 2005, la classe **ControlCollection** remplace la collection **Controls**. Les formulaires possèdent une classe **ControlCollection** par défaut à laquelle il est possible d'accéder à l'aide de la syntaxe `Me.Controls`.

Add, méthode

Dans Visual Basic 6.0, la méthode **Add** de la collection **Controls** est à liaison tardive ; les contrôles sont créés dans la méthode **Add** en spécifiant la classe **Control** comme argument.

Dans Visual Basic 2005, la méthode Add de la classe **ControlCollection** requiert que les contrôles aient été créés à l'aide du mot clé **New** avant d'être ajoutés à la collection.

Remove, méthode

La méthode **Remove** de la collection **Controls** de Visual Basic 6.0 pouvait seulement être utilisée pour des contrôles qui étaient ajoutés à l'aide de la méthode **Add** ; la classe **ControlCollection** de Visual Basic 2005 n'impose plus cette restriction.

Contrôles Timer et Menu

Dans Visual Basic 6.0, les contrôles **Timer** et **Menu** sont des membres de la collection **Controls**. Dans Visual Basic 2005, ces contrôles sont remplacés par Timer et MainMenu ou par les composants ContextMenu ; les composants ne sont pas des membres de la classe **ControlCollection**.

Contrôles contenus

La collection **Controls** de Visual Basic 6.0 inclut des contrôles qui sont des enfants d'un contrôle conteneur (par exemple, les contrôles situés dans un contrôle **Frame**) ; ce n'est plus vrai pour la classe **ControlCollection** de Visual Basic 2005. Pour parcourir tous les contrôles sur un formulaire, vous devez parcourir de manière récursive la classe **Controls** de chaque contrôle conteneur.

Modifications du code de la collection Controls

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour ajouter et supprimer des contrôles

Le code suivant illustre les différences entre la collection **Controls** de Visual Basic 6.0 et la classe **ControlCollection** de Visual Basic 2005.

```
' Visual Basic 6.0
Private Sub Command1_Click()
    ' Declare a new Control variable.
    Dim c As Control
    ' Create and add the new control.
    Set c = Controls.Add("VB.TextBox", "Text1")
    ' Make the new control visible.
    c.Visible = True
    ' Set the initial text.
    c.Text = "Hello"
    ' Retrieve the text from the new TextBox.
    If Controls.Count > 1 Then
        MsgBox (Controls("Text1").Text)
    End If
    ' Remove the new control.
    Controls.Remove (Text1)
    ' The following line causes a compilation error.
    ' You cannot remove controls added at design time.
    Controls.Remove (Command1)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handl
es Button1.Click
    ' Create a new TextBox control.
    Dim TextBox1 As New System.Windows.Forms.TextBox
    TextBox1.Name = "TextBox1"
    ' Add the new control to the form's Controls collection.
    Me.Controls.Add(TextBox1)
    ' No need to set Visible property.
    ' Set the initial text.
    TextBox1.Text = "Hello"
    ' Retrieve the text from the new TextBox.
    If Me.Controls.Count > 1 Then
        MsgBox(Me.Controls("TextBox1").Text)
    End If
    ' Remove the new control.
    Me.Controls.Remove(TextBox1)
    ' Remove the control added at design time.
    Me.Controls.Remove(Button1)
End Sub
```

Modifications du code pour parcourir la collection Controls

Le code suivant illustre une fonction permettant de parcourir tous les contrôles sur un formulaire, puis d'effacer tous les contrôles CheckBox. Cet exemple suppose que les contrôles **CheckBox** sont situés sur les contrôles GroupBox ou Panel plutôt que sur le formulaire. Dans l'exemple Visual Basic 2005, la collection Controls du formulaire incluant uniquement des contrôles situés directement sur le formulaire, la fonction s'appelle elle-même de manière récursive pour tout contrôle qui a des enfants.

```
' Visual Basic 6.0
Private Sub ClearChecks()
    For Each Control in Me.Controls
        If TypeOf Control Is CheckBox Then
            Control.Value = vbUnchecked
        End If
    Next
End Sub
```

VB

```
' Visual Basic 2005
Private Sub ClearChecks(ByVal Container As Control)
    Dim ctl As Control
    Dim chk As CheckBox
    For Each ctl In Container.Controls
        If TypeOf ctl Is CheckBox Then
            chk = ctl
            chk.Checked = False
        End If
        ' Recursively call this function for any container controls.
        If ctl.HasChildren Then
            ClearChecks(ctl)
        End If
    Next
End Sub
```

Mettre à niveau des remarques

Du fait des différences entre les collections **Controls** de Visual Basic 6.0 et de Visual Basic 2005, les appels à la méthode **Add** ne sont pas mis à niveau. Vous devez ajouter du code afin de recréer le comportement de votre application en utilisant la méthode **Add**.

Voir aussi

Référence

Control.Controls Property

Form.ControlCollection Class

Contrôle de données pour les utilisateurs de Visual Basic 6.0

Le contrôle **Data** de Visual Basic 6.0 est utilisé comme un mécanisme qui lie les contrôles à une base de données à l'aide de DAO. Visual Basic 2005 n'a aucun équivalent de contrôle **Data** ; l'architecture de liaison de données a changé et DAO n'est plus pris en charge. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.

Le contrôle **Data** fournit également une interface pour naviguer dans les données, avec des boutons permettant d'aller et venir d'une ligne à l'autre dans une table de base de données. Visual Basic 2005 a un contrôle équivalent, le contrôle BindingNavigator, qui contient également des boutons permettant d'ajouter et de supprimer des lignes.

Voir aussi

Référence

Vue d'ensemble du contrôle BindingNavigator (Windows Forms)

Concepts

Points à considérer avant d'effectuer la mise à niveau

Accès aux données pour les utilisateurs de Visual Basic 6.0

Contrôle DataGrid pour les utilisateurs de Visual Basic 6.0

Le contrôle **DataGrid** de Visual Basic 6.0 est remplacé par le contrôle DataGridView Windows Forms dans Visual Basic 2005. Les noms de certains événements, propriétés, méthodes et constantes ont changé. En outre, dans certains cas, leur comportement est également différent.

Différences conceptuelles

Liaison de données

Le contrôle Visual Basic 2005 **DataGridView** ne requiert ni méthode, ni événement spécifique aux données, car toutes les actions sont réalisées via la source de données. En raison de cette séparation entre la fonctionnalité des données et la présentation, la source de données peut être modifiée avec ou sans action dans l'interface utilisateur. En outre, plusieurs contrôles liés à la même source de données seront toujours synchronisés.

Navigation

Les propriétés pour l'affichage et la navigation présentes dans le contrôle **DataGrid** (notamment **TabAction**, **EnterAction**, **AllowArrows**, **WrapCellPointer** et **Scrollable**) ne sont plus nécessaires. Par exemple, la grille fonctionne comme si la propriété **Scrollable** avait pour valeur **True** : si la quantité de données est telle qu'elles ne peuvent pas s'afficher intégralement, une barre de défilement est ajoutée automatiquement. Par défaut, la navigation dans la grille est similaire à celle qui est proposée par Excel : l'utilisateur peut avancer à l'aide la touche TAB et reculer à l'aide de la combinaison de touches MAJ+TAB. Pour plus d'informations, consultez
Gestion par défaut du clavier et de la souris dans le contrôle DataGridView Windows Forms.

Propriété Caption

Dans Visual Basic 6.0, la propriété **Caption** est utilisée pour afficher une barre de titre au-dessus de la grille ; si la propriété **Caption** n'est pas définie, aucune barre de titre ne s'affiche.

Le contrôle **DataGridView** Visual Basic 2005 ne prend pas en charge de barre de titre. Toutefois, vous pouvez obtenir le même effet à l'aide d'un contrôle Label.

Mise en forme de données

Dans Visual Basic 6.0, la mise en forme des données dans le contrôle **DataGrid** est gérée à l'aide de la propriété **DataFormat** et d'un objet **StdDataFormat**. La mise en forme s'applique une colonne après l'autre.

Dans le contrôle Visual Basic 2005 **DataGridView**, la mise en forme s'effectue à l'aide de la propriété Format d'un objet DataGridViewCellStyle. La mise en forme peut s'appliquer individuellement aux cellules, colonnes ou lignes. Pour plus d'informations, consultez Mise en forme de données dans le contrôle DataGridView Windows Forms.

Propriété hWndEditor

Dans Visual Basic 6.0, la propriété **hWndEditor** est utilisée pour passer le handle de fenêtre affecté à la fenêtre de modification d'un contrôle **DataGrid** à un appel API Windows.

Le contrôle Visual Basic 2005 **DataGridView** n'a pas de handle de fenêtre distinct en mode édition ; à la place, utilisez la propriété Handle du contrôle **DataGridView**, ou tout contrôle d'édition incorporé.

Propriété MarqueeStyle

Dans Visual Basic 6.0, la propriété **MarqueeStyle** contrôle l'aspect d'une cellule ou d'une ligne sélectionnée, en modifiant le style de bordure, en inversant les couleurs de premier plan et d'arrière-plan, ou en appelant une fenêtre de modification.

Il n'y a aucun équivalent direct pour le contrôle Visual Basic 2005 **DataGridView**. Toutefois, vous pouvez obtenir le même effet à l'aide d'une combinaison des propriétés SelectionMode, CellBorderStyle et **Format**. Pour plus d'informations, consultez Modes de sélection dans le contrôle DataGridView Windows Forms.

Propriétés SelLength, SelStart et SelText

Dans le contrôle Visual Basic 6.0 **DataGrid**, lorsqu'une cellule entre en mode édition, les propriétés **SelLength**, **SelStart** et **SelText** permettent de définir la position initiale du signe insertion ou de mettre en surbrillance une partie du texte dans la cellule.

Dans le contrôle Visual Basic 2005 **DataGridView**, ces propriétés n'existent plus. Les cellules du contrôle **DataGridView** reposent sur le contrôle TextBox ; en ajoutant un code au gestionnaire d'événements EditingControlShowing, vous pouvez accéder aux propriétés SelectionLength, SelectionStart et SelectedText du contrôle sous-jacent.

Affichages fractionnés

Le contrôle Visual Basic 6.0 **DataGrid** prend en charge l'affichage fractionné, qui permet à l'utilisateur d'afficher les mêmes données côte à côte. L'objet **Split** et les propriétés **Split**, **Splits** et **TabAcrossSplits** contrôlent la capacité d'afficher un affichage fractionné.

Dans le contrôle Visual Basic 2005 **DataGridView**, ces propriétés n'existent plus. Toutefois, vous pouvez obtenir le même effet à l'aide d'un ou plusieurs contrôles SplitContainer et de plusieurs contrôles **DataGridView**. Pour dupliquer les fonctionnalités de la propriété **TabAcrossSplits**, vous pouvez utiliser la propriété StandardTab du contrôle **DataGridView**.

Modifications apportées au code du contrôle DataGrid

Le code suivant montre les différences entre Visual Basic 6.0 et Visual Basic 2005 en montrant comment, dans chaque version, mettre en surbrillance le texte dans une cellule lorsqu'un utilisateur sélectionne la cellule dans un contrôle **DataGridView**.

```
' Visual Basic 6.0
Private Sub DataGrid1_Click()
    DataGrid1.SelStart = 1
    DataGrid1.SelLength = DataGrid1.Text
    MsgBox("The selected text is " & DataGrid1.SelText)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub DataGridView1_EditingControlShowing( _
ByVal sender As Object, ByVal e As System.Windows.Forms. _
DataGridViewEditingControlShowingEventArgs) _
Handles DataGridView1.EditingControlShowing
    CType(e.Control, TextBox).SelectionStart = 0
    CType(e.Control, TextBox).SelectionLength = Len(CType(e.Control, _
      TextBox).Text)
    MsgBox("The selected text is " & CType(e.Control, _
      TextBox).SelectedText)
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle DataGrid

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
AddNewMode	Nouvelle implémentation. Les opérations sur les données sont gérées dans la source de données. Pour plus d'informations, consultez Affichage des données dans le contrôle DataGridView Windows Forms.
Align	Dock et Anchor
AllowAddNew	AllowNew (**BindingSource**)
AllowArrows	N'est plus une propriété ajustable ; la navigation au moyen des flèches est toujours autorisée. Pour plus d'informations, consultez Gestion par défaut du clavier et de la souris dans le contrôle DataGridView Windows Forms.
AllowDelete	AllowRemove (**BindingSource**)

AllowRowSizing	AllowUserToResizeRows
AllowUpdate	AllowEdit (**BindingSource**)
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
ApproxCount	Nouvelle implémentation. Les opérations sur les données sont gérées dans la source de données. Pour plus d'informations, consultez Affichage des données dans le contrôle DataGridView Windows Forms.
BackColor	BackgroundColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Bookmark	Nouvelle implémentation. Vous pouvez désormais accéder directement à tous les éléments.
Caption	Nouvelle implémentation. Utilisez un contrôle **Label** pour simuler une barre de titre.
Col	SelectedColumns
ColumnHeaders	ColumnHeadersVisible
Container	Parent ; hérité de **Control**.
CurrentCellModified	IsCurrentCellDirty, IsCurrentCellInEditMode
CurrentCellVisible	CurrentCell **Remarque** Si la propriété **CurrentCell** a pour valeur une cellule qui n'est pas visible, la grille défile jusqu'à cette cellule (comme en définissant **CurrentCellVisible** = **True**).
DataChanged	**IsCurrentCellDirty**, IsCurrentRowDirty
DataFormats	**DataGridViewCellStyle**.object. **Remarque** La mise en forme peut être définie individuellement pour une cellule, une colonne ou une ligne.
DefColWidth	Width Objet **DataGridViewColumn**.

DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
EditActive	**IsCurrentCellInEditMode**
FirstRow	FirstDisplayedScrollingRowIndex
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
HeadFont	Objet **DataGridViewCellStyle**. **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
HeadLines	Pas d'équivalent direct. Utilisez la propriété WrapMode de l'objet **DataGridViewCellStyle**, combinée à ColumnHeadersHeight.
Height	Height, hérité de la classe **Control**. **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
hWnd	**Handle**

hWndEditor	Nouvelle implémentation. Utilisez **Handle**.
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left, hérité de la classe **Control**. **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
LeftCol	FirstDisplayedScrollingColumnIndex
MarqueeStyle	Pas d'équivalent direct. Utilisez les propriétés **SelectionMode**, **CellBorderStyle** et **Format**.
RecordSelectors	RowHeadersVisible
Row	SelectedRows
RowDividerStyle	Propriétés GridColor, **CellBorderStyle**, RowHeadersBorderStyle et ColumnHeadersBorderStyle.
RowHeight	Height
SelBookmarks	Nouvelle implémentation. Vous pouvez désormais accéder directement à tous les éléments.
SelEndCol **SelStartCol**	SelectedCells, **SelectedColumns**
SelLength	Pas d'équivalent direct. Utilisez la propriété **SelectionLength** dans le gestionnaire d'événements **EditingControlShowing**.
SelStart	Pas d'équivalent direct. Utilisez la propriété **SelectionStart** dans le gestionnaire d'événements **EditingControlShowing**.
SelText	Pas d'équivalent direct. Utilisez la propriété **SelectedText** dans le gestionnaire d'événements **EditingControlShowing**.
Split **Splits** **TabAcrossSplits**	Nouvelle implémentation. Les affichages fractionnés ne sont pas directement pris en charge ; utilisez un contrôle **SplitContainer**.

TabAction	**StandardTab**
Tag	Nouvelle implémentation.
Text	**CurrentCell.Value** **Remarque** La propriété **Value** retourne un **Object** ; utilisez **CStr** ou **ToString** pour le convertir en un **String**.
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
VisibleCols	Méthode **DisplayedColumnCount**. **Remarque** Dans Visual Basic 6.0, les colonnes partiellement visibles sont systématiquement incluses dans le compte ; la méthode **DisplayedColumnCount** utilise un paramètre **includePartialColumns** de type Boolean pour déterminer si les colonnes partiellement visibles sont incluses ou non.
VisibleRows	Méthode **DisplayedRowCount**. **Remarque** Dans Visual Basic 6.0, les lignes partiellement visibles sont systématiquement incluses dans le compte ; la méthode **DisplayedRowCount** utilise un paramètre **includePartialRows** de type Boolean pour déterminer si les lignes partiellement visibles sont incluses ou non.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, hérité de la classe **Control**. **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WrapCellPointer	Cette propriété n'est plus ajustable ; le comportement par défaut est **WrapCellPointer** = **True**.

Méthodes

Visual Basic 6.0	Équivalent Visual Basic 2005
CaptureImage	Nouvelle implémentation. La capture du contenu d'un contrôle **DataGridView** dans un contrôle **PictureBox** n'est pas prise en charge.

ClearFields	Nouvelle implémentation. La mise en forme de colonne est automatique en cas de renouvellement de liaison.
ClearSelCols	ClearSelection
ColContaining	IndexOf (**DataGridViewColumnCollection**)
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GetBookmark	Nouvelle implémentation. Les signets ne sont plus pris en charge.
HoldFields	Nouvelle implémentation. La mise en forme de colonne est conservée en cas de renouvellement de liaison.
Move	SetBounds, hérité de la classe **Control**. **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Rebind	ResetBindings, hérité de la classe **Control**.
RowBookmark	Nouvelle implémentation. Les signets ne sont plus pris en charge.
RowContaining	IndexOf (**DataGridViewColumnCollection**)
RowTop	GetContentBounds (**DataGridViewCell**)
Scroll	Nouvelle méthode d'implémentation. Utilisez la propriété **CurrentCell**.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
SplitContaining	Nouvelle implémentation. Les affichages fractionnés ne sont pas directement pris en charge ; utilisez un contrôle **SplitContainer**.
ZOrder	Fonctions **BringToFront()** ou **SendToBack()**

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005**
AfterColEdit	CellEndEdit

AfterColUpdate **AfterDelete**	Nouvelle implémentation. Les opérations sur les données sont gérées dans la source de données. Pour plus d'informations, consultez Affichage des données dans le contrôle DataGridView Windows Forms.
AfterUpdate	RowsAdded
BeforeColEdit	CellBeginEdit
BeforeColUpdate **BeforeDelete** **BeforeInsert** **BeforeUpdate**	Nouvelle implémentation. Les opérations sur les données sont gérées dans la source de données. Pour plus d'informations, consultez Affichage des données dans le contrôle DataGridView Windows Forms.
ButtonClick	Click (contrôle **Button**)
Change	TextChanged **Remarque** Le comportement de l'événement **TextChanged** est légèrement différent. Pour plus d'informations, consultez Événement Change du contrôle ComboBox pour les utilisateurs de Visual Basic 6.0.
Click	SelectedIndexChanged
ColEdit	**CellBeginEdit**
ColResize	ColumnWidthChanged
DblClick	CellMouseDoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Error	DataError
HeadClick	ColumnHeaderMouseClick
MouseDown	CellMouseDown
MouseMove	CellMouseMove
MouseUp	CellMouseUp

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
OnAddNew	**RowsAdded**
RowColChange	CurrentCellChanged
RowResize	RowHeightChanged
SelChange	SelectionChanged
SplitChange	Nouvelle implémentation. Les affichages fractionnés ne sont pas directement pris en charge ; utilisez un contrôle **SplitContainer**.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, l'événement **Change** du contrôle **DataGrid** est mappé à l'événement **TextChanged** du contrôle Visual Basic 2005 **DataGridView**. Le comportement de l'événement **TextChanged** diffère de celui de l'événement **Change** ; cette différence peut produire des résultats imprévus dans votre code.

Tout code relatif à des méthodes ou des événements spécifiques à des données n'est pas mis à niveau. Des commentaires d'avertissement sont ajoutés au code et le code doit être supprimé ou modifié avant la compilation de l'application.

Voir aussi

Référence

Vue d'ensemble du contrôle DataGridView (Windows Forms)

Comparaison des contrôles et des objets programmables dans différents langages et dans différentes bibliothèques

Concepts

Résumé de la technologie du contrôle DataGridView (Windows Forms)

Nouveautés dans les Windows Forms et les contrôles Windows Forms

Autres ressources

DataGridView, contrôle (Windows Forms)

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Liaison de données Windows Forms

Contrôle DirListBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **DirListBox** de Visual Basic 6.0 a été rendu obsolète par les composants OpenFileDialog et SaveFileDialog dans Visual Basic 2005.

Différences conceptuelles

Le contrôle **DirListBox** de Visual Basic 6.0 était principalement utilisé pour afficher des répertoires et des chemins d'accès dans une boîte de dialogue Fichier Ouvrir ou Enregistrer.

Dans Visual Basic 2005, les **OpenFileDialog** Windows Forms et les composants **SaveFileDialog** offrent la possibilité de créer des boîtes de dialogue Windows standard pour travailler avec les fichiers, qui dispensent, dans la plupart des cas, de l'utilisation du contrôle **DirListBox**.

Remarque
Nous vous recommandons d'utiliser les composants **OpenFileDialog** et **SaveFileDialog** pour offrir une expérience cohérente et familière. Si vous pensez qu'il est nécessaire de créer vos propres boîtes de dialogue Fichier, Visual Basic 2005 fournit un contrôle **DirListBox** dans le cadre de la bibliothèque Runtime de compatibilité Microsoft Visual Basic. Pour plus d'informations, consultez Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application.

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 vers Visual Basic 2005, tout contrôle **DirListBox** existant est mis à niveau vers le contrôle **VB6.DirListBox** qui est fourni dans le cadre de la bibliothèque de compatibilité (**Microsoft.VisualBasic.Compatibility**).

Voir aussi

Tâches

Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application

Référence

Vue d'ensemble du composant OpenFileDialog (Windows Forms)

Vue d'ensemble du composant SaveFileDialog (Windows Forms)

Contrôle DriveListBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **DriveListBox** de Visual Basic 6.0 a été rendu obsolète par les composants OpenFileDialog et SaveFileDialog dans Visual Basic 2005.

Différences conceptuelles

Le contrôle **DriveListBox** de Visual Basic 6.0 était utilisé en général pour afficher une liste de lecteurs dans une boîte de dialogue **Ouvrir** ou **Enregistrer**.

Dans Visual Basic 2005, les composants **OpenFileDialog** et **SaveFileDialog** offrent la possibilité de créer des boîtes de dialogue Windows standard pour travailler avec les fichiers, qui dispensent, dans la plupart des cas, de l'utilisation du contrôle **DriveListBox**.

Remarque
L'utilisation des composants **OpenFileDialog** et **SaveFileDialog** crée chez l'utilisateur une impression cohérente et familièr e. Si vous avez besoin de créer vos propres boîtes de dialogue Fichier, Visual Basic 2005 fournit un contrôle **DriveListBox** da ns le cadre de la bibliothèque Runtime de compatibilité Microsoft Visual Basic. Pour plus d'informations, consultez Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application.

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 vers Visual Basic 2005, tout contrôle **DriveListBox** existant est mis à niveau vers le contrôle **VB6.DriveListBox** de la bibliothèque **Microsoft.VisualBasic.Compatibility**.

Voir aussi

Tâches

Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application

Référence

Vue d'ensemble du composant OpenFileDialog (Windows Forms)

Vue d'ensemble du composant SaveFileDialog (Windows Forms)

Contrôle FileListBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **FileListBox** de Visual Basic 6.0 a été rendu obsolète par les composants OpenFileDialog et SaveFileDialog dans Visual Basic 2005.

Différences conceptuelles

Le contrôle **FileListBox** Visual Basic 6.0 était principalement utilisé pour sélectionner des fichiers dans une boîte de dialogue **Ouvrir** ou **Enregistrer**.

Dans Visual Basic 2005, les **OpenFileDialog** Windows Forms et les composants **SaveFileDialog** offrent la possibilité de créer des boîtes de dialogue Windows standard pour travailler avec les fichiers, qui dispensent, dans la plupart des cas, de l'utilisation du contrôle **FileListBox**.

Remarque

Nous vous recommandons d'utiliser les composants **OpenFileDialog** et **SaveFileDialog** pour offrir une expérience cohérente et familière. Si vous pensez qu'il est nécessaire de créer vos propres boîtes de dialogue Fichier, Visual Basic 2005 fournit un contrôle **FileListBox** dans le cadre de la bibliothèque Runtime de compatibilité Microsoft Visual Basic. Pour plus d'informations, consultez Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application.

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 vers Visual Basic 2005, tout contrôle **FileListBox** existant est mis à niveau vers le contrôle **VB6.FileListBox**, qui est fourni dans le cadre de la bibliothèque Microsoft.VisualBasic.Compatibility.

Voir aussi

Tâches

Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application

Référence

Vue d'ensemble du composant OpenFileDialog (Windows Forms)

Vue d'ensemble du composant SaveFileDialog (Windows Forms)

Objet Font pour les utilisateurs de Visual Basic 6.0

Visual Basic 6.0 offre deux moyens de définir des propriétés **Font** ; dans Visual Basic 2005 il existe une seule classe Font.

Différences conceptuelles

Dans Visual Basic 6.0, les propriétés **Font** peuvent être définies par le biais de l'objet **stdFont**, plutôt que directement. L'objet **stdFont** Visual Basic 6.0 est remplacé dans Visual Basic 2005 par la classe **Font** ; il n'y a aucun mappage direct des propriétés et méthodes. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.

Voir aussi

Concepts

Gestion des polices pour les utilisateurs de Visual Basic 6.0

Objet Form pour les utilisateurs de Visual Basic 6.0

L'objet **Form** dans Visual Basic 6.0 est remplacé par la classe Form dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, leur comportement est également différent.

Différences conceptuelles

Événements Activate et Deactivate

Dans Visual Basic 6.0, les événements **Activate** et **Deactivate** sont déclenchés uniquement lors du basculement entre formulaires.

Dans Visual Basic 2005, les événements Activated et Deactivate sont également déclenchés lors du basculement de ou vers d'autres applications.

Événement QueryUnload

Dans Visual Basic 6.0, l'événement **QueryUnload** accepte deux arguments, *Cancel* et *UnloadMode*. En interrogeant l'argument *UnloadMode*, vous pouvez déterminer la raison pour laquelle le formulaire est déchargé, puis annuler si nécessaire.

Dans Visual Basic 2005, l'événement **QueryUnload** est remplacé par l'événement FormClosing ; *UnloadMode* est remplacé par CloseReason.

Propriété Picture

Dans Visual Basic 6.0, une image assignée à la propriété **Picture** est affichée dans le coin supérieur gauche du formulaire si la bitmap est plus petite que le formulaire.

Dans Visual Basic 2005, la propriété **Picture** est remplacée par la propriété BackgroundImage ; si l'image assignée à la propriété **BackgroundImage** est plus petite que le formulaire, elle sera disposée en mosaïque par défaut.

Propriété Moveable

Dans Visual Basic 6.0, la propriété **Moveable** d'un formulaire peut avoir la valeur **False** pour empêcher qu'un utilisateur ne déplace le formulaire au moment de l'exécution. Les Windows Forms dans Visual Basic 2005 n'ont pas de propriété équivalente.

Bien que cela soit généralement considéré comme une mauvaise conception d'interface utilisateur, vous pouvez parvenir à un comportement similaire dans Visual Basic 2005 en affectant à la propriété FormBorderStyle la valeur **None** et à la propriété ControlBox la valeur **False**.

Méthode ValidateControls

Dans Visual Basic 6.0, la méthode **ValidateControls** est utilisée pour forcer l'événement **Validate** du contrôle qui a le focus lors de la fermeture d'un formulaire ; si l'événement **Validate** échoue, une erreur est déclenchée.

Dans Visual Basic 2005, la méthode **ValidateControls** est remplacée par la méthode Validate, qui retourne **True** ou **False**.

Événements de souris pour les formulaires MDI

Dans Visual Basic 6.0, les formulaires MDI prennent en charge les événements de souris. Dans Visual Basic 2005, les événements **Click**, **MouseDown**, **MouseMove** et **MouseUp** ne sont plus pris en charge pour les formulaires MDI, parce que ceux-ci ne disposent pas d'une zone client pour la réception des événements de souris.

Modifications apportées au code de l'objet Form

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code de détermination de la raison de la fermeture d'un formulaire

Le code suivant peut vous aider à déterminer la raison de la fermeture d'un formulaire, puis comment agir selon la raison. Dans l'exemple Visual Basic 6.0, l'argument *UnloadMode* de l'événement **QueryUnload** est utilisé. Dans l'exemple Visual Basic 2005, l'argument est remplacé par le paramètre **CloseReason** dans le gestionnaire d'événements **FormClosing**.

```
' Visual Basic 6.0
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If UnloadMode = vbFormControlMenu Then
        Cancel = True
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_FormClosing(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles _
MyBase.FormClosing
    If e.CloseReason.UserClosing Then
        e.Cancel = True
    End If
End Sub
```

Modifications du code de validation d'un contrôle à la fermeture d'un formulaire

Le code suivant montre comment forcer la validation du contrôle qui a le focus à la fermeture du formulaire. Cet exemple suppose que la propriété **CausesValidation** du contrôle **TextBox** a la valeur par défaut **True**.

```
' Visual Basic 6.0
Private Sub Text1_Validate(Cancel As Boolean)
    If Text1.Text = "" Then
        MsgBox ("Please enter a name")
        Cancel = True
End If

Private Sub Form_Unload(Cancel As Integer)
    OnError GoTo ERR_HANDLER
    Me.ValidateControls

ERR_HANDLER:
    ' If validation failed cancel the Unload event.
    If Err.Number = 380 Then
        Cancel = True
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBox1_Validating(ByVal sender As Object, ByVal e As _
System.ComponentModel.CancelEventArgs) Handles TextBox1.Validating
    If TextBox1.Text = "" Then
        MsgBox("Please enter a name")
        e.Cancel = True
    End If
End Sub
```

VB

```
Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e _
As System.Windows.Forms.FormClosingEventArgs) Handles _
MyBase.FormClosing
    ' If validation failed cancel the Closing event.
    If Me.Validate = False Then
        e.Cancel = True
    End If
End Sub
```

Équivalences des propriétés, méthodes et événements de l'objet Form

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs propriétés Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés de l'objet Form

Visual Basic 6.0	**Équivalent Visual Basic 2005**
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
AutoRedraw	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
BorderStyle	**FormBorderStyle**
Caption	Text
ClipControls	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Controls	Controls **Remarque** La collection **Controls** dans Visual Basic 2005 présente des différences. Pour plus d'informations, consultez Collection Controls pour les utilisateurs de Visual Basic 6.0.
Count	Count **Remarque** La collection **Controls** dans Visual Basic 2005 présente des différences. Pour plus d'informations, consultez Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0.
CurrentX **CurrentY**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
DrawMode **DrawStyle** **DrawWidth**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
FillColor **FillStyle**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
HasDC	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
HDC	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Image	Nouvelle implémentation. Dans Visual Basic 6.0, la propriété **Image** retournait un handle vers une bitmap ; dans Visual Basic 2005, les images n'ont pas de handles.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
LinkMode **LinkTopic**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
MaxButton	MaximizeBox

MDIChild	MdiParent **Remarque** Le comportement MDI est différent dans Visual Basic 2005. Pour plus d'informations, consultez MDI pour les utilisateurs de Visual Basic 6.0.
MinButton	MinimizeBox
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
Moveable	Nouvelle implémentation. Pour plus d'informations, consultez Propriété Moveable pour les utilisateurs de Visual Basic 6.0.
Name	Name
NegotiateMenus	Nouvelle implémentation. Pour plus d'informations, consultez Objet Menu pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Palette **PaletteMode**	Nouvelle implémentation. Pour plus d'informations, consultez Palettes pour les utilisateurs de Visual Basic 6.0.
Picture	**BackgroundImage** **Remarque** Dans Visual Basic 6.0, le **Picture** était affiché dans le coin supérieur gauche du formulaire ; dans Visual Basic 2005, **BackgroundImage** est disposé en mosaïque.
RightToLeft: **True** **False**	RightToLeft Valeur d'énumération Yes
ScaleHeight **ScaleLeft** **ScaleMode** **ScaleTop** **ScaleWidth**	Nouvelle implémentation. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

StartUpPosition: **0 – Manual** **1 – CenterOwner** **2 – CenterScreen** **3 – Windows Default**	StartPosition Valeur d'énumération Manual Valeur d'énumération CenterParent Valeur d'énumération CenterScreen Valeur d'énumération WindowsDefaultLocation
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisButton	HelpButton **Remarque** Le comportement de l'aide dans Visual Basic 2005 présente des différences. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelp	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes de l'objet Form

Visual Basic 6.0	Équivalent Visual Basic 2005
Circle	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Cls	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Line	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

PaintPicture	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Point	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
PopupMenu	Nouvelle implémentation. Pour plus d'informations, consultez Objet Menu pour les utilisateurs de Visual Basic 6.0.
PrintForm	Nouvelle implémentation. Pour plus d'informations, consultez Modifications de l'impression pour les utilisateurs de Visual Basic 6.0.
Pset	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Scale **ScaleX** **ScaleY**	Nouvelle implémentation. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
SetFocus	Activate
Show	Show ou ShowDialog
TextHeight	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
TextWidth	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
ValidateControls	Validate **Remarque** La méthode **Validate** retourne **True** ou **False** ; **ValidateControls** a déclenché une erreur si la validation a échoué.
WhatsThisMode	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	Fonction BringToFront ou SendToBack **BringToFront** **SendToBack**

Événements de l'objet Form

Visual Basic 6.0	**Équivalent Visual Basic 2005**
Activate	**Activated** **Remarque** Dans Visual Basic 6.0, l'événement **Activate** est déclenché uniquement lors du basculement entre les formulaires de l'application ; dans Visual Basic 2005, l'événement **Activated** est déclenché également lors du basculement à partir d'autres applications.

Click	Click **Remarque** L'événement **Click** n'est pas pris en charge dans les formulaires MDI.
DblClick	DoubleClick
Deactivate	**Deactivate** **Remarque** Dans Visual Basic 6.0, l'événement **Deactivate** est déclenché uniquement lors du basculement entre les formulaires de l'application ; dans Visual Basic 2005, il est également déclenché lors du basculement vers d'autres applications.
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
Initialize	Méthode **New** **Remarque** Le comportement de la méthode **New** diffère de celui de l'événement **Initialize**. Pour plus d'informations, consultez Tâches des formulaires pour les utilisateurs de Visual Basic 6.0.
LinkClose **LinkError** **LinkExecute** **LinkOpen**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
MouseDown	MouseDown **Remarque** L'événement **MouseDown** n'est pas pris en charge dans les formulaires MDI.

MouseMove	MouseMove **Remarque** L'événement **MouseMove** n'est pas pris en charge dans les formulaires MDI.
MouseUp	MouseUp **Remarque** L'événement **MouseUp** n'est pas pris en charge dans les formulaires MDI.
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
QueryUnload	**FormClosing** **Remarque** Dans Visual Basic 6.0, **QueryUnload** accepte deux arguments, *Cancel* et *UnloadMode*. Dans Visual Basic 2005, *Cancel* est remplacé par Cancel ; *UnloadMode* est remplacé par **CloseReason**.
Terminate	Méthode Dispose **Remarque** Le comportement de la méthode **Dispose** diffère de celui de l'événement **Terminate**. Pour plus d'informations, consultez Événements de formulaire pour les utilisateurs de Visual Basic 6.0.
Unload	**FormClosing** **Remarque** Le comportement de l'événement **FormClosing** diffère de celui de l'événement **Unload**. Pour plus d'informations, consultez Événements de formulaire pour les utilisateurs de Visual Basic 6.0.

Voir aussi

Concepts

Tâches des formulaires pour les utilisateurs de Visual Basic 6.0

Autres ressources

Windows Forms

Collection Forms pour les utilisateurs de Visual Basic 6.0

La collection **Forms** de Visual Basic 6.0 est remplacée par la propriété My.Application.OpenForms dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, la collection **Forms** est une collection dont les éléments représentaient chaque formulaire chargé dans une application. La collection inclut le formulaire MDI de l'application, formulaires enfants MDI et les formulaires non-MDI. La collection **Forms** a une propriété unique, **Count**, qui spécifie le nombre d'éléments dans la collection.

La propriété OpenForms de Visual Basic 2005 retourne un objet FormCollection qui contient tous les formulaires ouverts de l'application. Le comportement est identique à celui de la collection **Forms**, et il peut être utilisé de la même manière.

Modifications du code de la collection Forms

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour obtenir un compte de formulaires ouverts

Le code suivant montre comment retourner le nombre de formulaires actuellement ouverts dans une application.

```
' Visual Basic 6.0
MsgBox Forms.Count
```

VB

```
' Visual Basic 2005
MsgBox(CStr(My.Application.OpenForms.Count))
```

Modifications du code pour définir les propriétés de tous les formulaires ouverts

Le code suivant montre comment modifier le titre de chaque formulaire ouvert dans une application.

```
' Visual Basic 6.0
For Each Form in Forms
    Forms(i).Caption = "Hello"
Next
```

VB

```
' Visual Basic 2005
For Each f As Form In My.Application.OpenForms
    f.Text = "Hello"
Next
```

Voir aussi

Tâches

Comment : accéder à tous les formulaires ouverts d'une application

Référence

My.Application.OpenForms, propriété

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Contrôle Frame pour les utilisateurs de Visual Basic 6.0

Le contrôle **Frame** Visual Basic 6.0 est remplacé par deux contrôles dans Visual Basic 2005 : le contrôle GroupBox et le contrôle Panel.

Différences conceptuelles

Dans Visual Basic 6.0, le contrôle **Frame** sert de conteneur pour le groupement des contrôles. Dans Visual Basic 2005, le contrôle **Frame** est remplacé par le contrôle **GroupBox** ou le contrôle **Panel**.

Le contrôle **GroupBox** est équivalent à un contrôle **Frame** doté de la propriété **BorderStyle** définie avec la valeur **1 – Fixed Single**. Il possède une bordure visible et éventuellement une légende.

Le contrôle **Panel** est équivalent à un contrôle **Frame** doté de la propriété **BorderStyle** définie avec la valeur **0 – None**. Il ne possède ni bordure ni légende.

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris les différences relatives à la liaison de données, la gestion des polices, le glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0.

Équivalences des propriétés, méthodes et événements du contrôle Frame

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés du contrôle Frame

Visual Basic 6.0	Équivalent Visual Basic 2005
Appearance	Contrôle FlatStyle (**GroupBox** uniquement)
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
BorderStyle	Contrôle BorderStyle (**GroupBox** uniquement)
Caption	Contrôle Text (**GroupBox** uniquement) **Remarque** Le contrôle **Panel** ne possède pas de propriété **Text**.
ClipControls	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Container	Parent
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
RightToLeft	RightToLeft

ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes du contrôle Frame

Visual Basic 6.0	Équivalent Visual Basic 2005
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	Fonction BringToFront ou SendToBack

Événements du contrôle Frame

Visual Basic 6.0	Équivalent Visual Basic 2005
Click	Click (**Panel** uniquement ; **GroupBox** n'a aucun équivalent)
DblClick	DoubleClick (**Panel** uniquement ; **GroupBox** n'a aucun équivalent)

DragDrop **DragOver** **OLECompleteDrag** **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, les contrôles **Frame** dont les propriétés **BorderStyle** ont la valeur **0 - None** sont mis à niveau vers les contrôles **Panel** ; tous les autres contrôles **Frame** sont mis à niveau vers les contrôles **GroupBox**.

Voir aussi

Référence

Vue d'ensemble du contrôle GroupBox (Windows Forms)

Concepts

Introduction au contrôle Panel

Contrôle HScrollBar pour les utilisateurs de Visual Basic 6.0

Le contrôle **HScrollBar** de Visual Basic 6.0 est remplacé par le contrôle HScrollBar Windows Forms dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Change, événement

Dans Visual Basic 6.0, lorsque la propriété **Value** d'un contrôle **HScrollBar** est modifiée, l'événement **Change** est déclenché.

Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché; l'événement ValueChanged est déclenché à la place.

Value, propriété

Dans Visual Basic 6.0, lorsque l'événement **Scroll** ou **Change** d'un contrôle **HScrollBar** est déclenché, la valeur de défilement actuelle est assignée à la propriété **Value**.

Lorsque l'événement Scroll ou **ValueChanged** est déclenché dans Visual Basic 2005, la propriété Value du contrôle n'est pas encore mise à jour. Si vous devez émuler le comportement de Visual Basic 6.0, vous pouvez créer une fonction d'assistance pour récupérer la valeur. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.

LargeChange, propriété

Dans Visual Basic 6.0, la propriété **LargeChange** détermine la quantité des modifications apportées au paramètre de la propriété **Value** dans un contrôle de barre de défilement lorsque l'utilisateur clique sur la zone entre la case de défilement et la flèche de défilement.

Dans Visual Basic 2005, la valeur par défaut de la propriété LargeChange est 10 ; dans Visual Basic 6.0, elle est égale à 1.

Max, propriété

Dans Visual Basic 6.0, la propriété **Max** détermine le paramètre de propriété **Value** maximal dans un contrôle de barre de défilement.

Dans Visual Basic 2005, la propriété **Max** est remplacée par la propriété Maximum ; la valeur par défaut est 100, dans Visual Basic 6.0, elle est égale à 32767.

Min, propriété

Dans Visual Basic 6.0, la propriété **Min** détermine le paramètre de propriété **Value** minimal dans un contrôle de barre de défilement. La valeur de la propriété **Min** peut être supérieure à celle de la propriété **Max**.

Dans Visual Basic 2005, la propriété **Min** est remplacée par la propriété Minimum, dont la valeur doit toujours être inférieure à celle de la propriété **Maximum**.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, la fonctionnalité glisser-déplacer, la prise en charge de l'aide, etc.

Équivalences des propriétés, méthodes et événements du contrôle HScrollBar

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Des liens nécessaires sont fournis pour expliquer des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés du contrôle HScrollBar

Visual Basic 6.0	Équivalent Visual Basic 2005
Container	Parent

DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
LargeChange	LargeChange **Remarque** La valeur par défaut dans Visual Basic 6.0 est 1 ; dans Visual Basic 2005, la valeur par défaut est 10.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Max	Maximum **Remarque** La valeur par défaut dans Visual Basic 6.0 est 32767 ; dans Visual Basic 2005, la valeur par défaut est 100.
Min	Minimum **Remarque** Dans Visual Basic 6.0, une valeur supérieure à **Max** peut être affectée à **Min** ; dans Visual Basic 2005, la valeur de la propriété **Minimum** ne peut pas être supérieure à celle de la propriété **Maximum**.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm

RightToLeft: **True** **False**	RightToLeft Valeur d'énumération Yes Valeur d'énumération No
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Value	**Value** **Remarque** Dans Visual Basic 6.0, lorsque **Value** est modifié, l'événement **Change** est déclenché. Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché ; utilisez l'événement **ValueChanged** à la place.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes du contrôle HScrollBar

Visual Basic 6.0	Équivalent Visual Basic 2005
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	Fonction BringToFront ou SendToBack **BringToFront** **SendToBack**

Événements du contrôle HScrollBar

Visual Basic 6.0	Équivalent Visual Basic 2005

Change	**ValueChanged** **Remarque** Le comportement des événements **Change** et **Scroll** a été modifié dans Visual Basic 2005. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
Scroll	**Scroll** **Remarque** Le comportement des événements **Change** et **Scroll** a été modifié dans Visual Basic 2005. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tout code dans les gestionnaires d'événements **Change** ou **Scroll** pour les contrôles **HScrollBar** ou **VScrollBar** est modifié en une procédure en raison d'une différence dans la séquence d'événements. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.

Voir aussi

Référence

Vue d'ensemble des contrôles HScrollBar et VScrollBar (Windows Forms)

Contrôle Image pour les utilisateurs de Visual Basic 6.0

Le contrôle **Image** dans Visual Basic 6.0 n'a aucun équivalent direct dans Visual Basic 2005. Le contrôle **Image** est un contrôle léger ; Visual Basic 2005 ne prend pas en charge les contrôles légers.

Mise à niveau de la propriété Stretch

Les propriétés, méthodes et événements du contrôle **Image** constituent un sous-ensemble de ceux du contrôle **PictureBox** (exception faite de la propriété **Stretch**). Pour plus d'informations, consultez
Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0.

Le tableau suivant affiche le mappage entre la propriété **Stretch** et l'équivalent Visual Basic 2005 du contrôle PictureBox.

Équivalents de la propriété Stretch

Visual Basic 6.0	Équivalent Visual Basic 2005
Stretch = **False**	PictureBoxSizeMode = Normal
Stretch = **True**	**PictureBoxSizeMode** = StretchImage

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **Image** sont mis à niveau vers des contrôles **PictureBox**.

Voir aussi

Référence

Vue d'ensemble du contrôle PictureBox (Windows Forms)

Concepts

Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0

Contrôle ImageList pour les utilisateurs de Visual Basic 6.0

Le contrôle **ImageList** de Visual Basic 6.0 est remplacé par le composant ImageList Windows Forms dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

BackColor

Dans Visual Basic 6.0, le contrôle **ImageList** a une propriété **BackColor**. Toutefois, il est sans effet, car le contrôle n'est pas visible au moment de l'exécution.

Dans Visual Basic 2005, le nouveau composant **ImageList** ne dispose plus d'une propriété **BackColor**.

Propriétés en haut, à gauche

Dans Visual Basic 6.0, le contrôle **ImageList** a des propriétés **Left** et **Top** qui définissent l'emplacement du contrôle sur un formulaire.

Dans Visual Basic 2005, le composant **ImageList** se trouve dans la **Barre d'état des composants** au lieu d'être sur le formulaire. Les propriétés **Left** ou **Top** sont donc inutiles.

Propriété UseMaskColor

Dans Visual Basic 6.0, la propriété **UseMaskColor** détermine si la couleur assignée dans la propriété **MaskColor** est utilisée comme un masque.

Dans Visual Basic 2005, le composant **ImageList** n'a plus de propriété **UseMaskColor**, et la propriété **MaskColor** est remplacée par une propriété **TransparentColor**. Pour empêcher la couleur définie dans la propriété **TransparentColor** d'être utilisée comme un masque, définissez-la en **Color.Transparent**.

Méthode Overlay

Dans Visual Basic 6.0, la méthode **Overlay** d'un contrôle **ImageList** vous permet de créer une troisième image composite à partir de deux objets **ListImage**. Par exemple, vous pouvez définir un masque pour une image de sorte qu'une partie de la deuxième image transparaisse.

Dans Visual Basic 2005, le composant **ImageList** ne prend plus en charge la méthode **Overlay**. Pour obtenir le même effet, vous devez créer l'image composite à l'aide d'un programme graphique, puis ajouter cette image à **ImageList.**

Modifications apportées au code du contrôle ImageList

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications apportées au code pour définir un MaskColor

Le code suivant montre comment spécifier une couleur à utiliser pour un masque et comment l'activer et le désactiver.

```
' Visual Basic 6.0
' Specify the color to be used as a mask.
ImageList1.MaskColor = vbWhite
' Use the mask.
ImageList1.UseMaskColor = True
' Don't use the mask.
ImageList1.UseMaskColor = False
```

VB

```
' Visual Basic 2005
' Specify the color to be used as a mask and use the mask.
ImageList1.TransparentColor = Color.White
' Don't use the mask.
ImageList1.TransparentColor = Color.Transparent
```

Équivalences des propriétés et des méthodes du contrôle ImageList

Les tableaux suivants répertorient les propriétés et les méthodes de Visual Basic 6.0, ainsi que leurs équivalents Visual Basic 2005. Les propriétés et les méthodes, qui ont des noms et des comportements identiques, ne sont pas répertoriées. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
BackColor	Nouvelle implémentation. **ImageList** est un composant.
hImageList	Handle
ImageHeight **ImageWidth**	ImageSize
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Nouvelle implémentation. **ImageList** est un composant.
ListImage	Image (classe **ImageCollection**)
ListImages	ImageCollection
MaskColor	TransparentColor
Parent	Méthode FindForm
Top	Nouvelle implémentation. **ImageList** est un composant.
UseMaskColor	Nouvelle implémentation. Affectez à **TransparentColor** la valeur **Transparent**.

Méthodes

Nom	**Équivalent Visual Basic 2005.**
Overlay	Nouvelle implémentation.

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, tous les contrôles **ImageList** sont mis à niveau vers des composants Windows Forms **ImageList**. Lorsque les propriétés, méthodes et événements n'ont aucun équivalent ou présentent des différences de comportement, des remarques ou avertissements de mise à niveau sont ajoutés à votre code.

Voir aussi

Référence

Vue d'ensemble du composant ImageList (Windows Forms)

Autres ressources

ImageList, composant (Windows Forms)

Contrôle Label pour les utilisateurs de Visual Basic 6.0

Cette rubrique compare le contrôle **Label** de Visual Basic 6.0 avec son équivalent dans Visual Basic 2005.

Le contrôle **Label** dans Visual Basic 6.0 est remplacé par le contrôle Label Windows Forms dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, leur comportement est également différent.

Différences conceptuelles

BackStyle, propriété

Dans Visual Basic 6.0, la propriété **BackStyle** du contrôle **Label** indique si l'étiquette a un arrière-plan transparent. L'affectation de la valeur **0 – Transparent** à la propriété **BackStyle** permet d'afficher une image d'arrière-plan à travers l'étiquette. Si l'étiquette est placée sur d'autres contrôles, les contrôles apparaissent également à travers celle-ci.

Dans Visual Basic 2005, la propriété **BackStyle** n'existe plus et le comportement de transparence est légèrement différent. Pour émuler le comportement transparent dans Visual Basic 2005, affectez la valeur Transparent à la propriété BackColor. Cela permet à l'image d'arrière-plan d'apparaître à travers l'étiquette. Si l'étiquette est placée sur d'autres contrôles, les contrôles apparaissent uniquement à travers celle-ci s'ils sont plus hauts dans l'ordre z.

Remarque

La méthode **ZOrder** n'existe plus dans Visual Basic 2005. Au moment du design, vous pouvez utiliser la commande **Mettre au premier plan** ou **Mettre en arrière-plan** dans le menu **Format** ; au moment de l'exécution, vous pouvez utiliser la fonction BringToFront ou SendToBack.

WordWrap, propriété

Dans Visual Basic 6.0, la propriété **WordWrap** détermine si le texte est renvoyé à la ligne sur plusieurs lignes lorsqu'il est trop long pour contenir dans une étiquette.

Dans Visual Basic 2005, le texte dans un contrôle **Label** est automatiquement renvoyé à la ligne. La seule façon d'empêcher le renvoi à la ligne est de faire en sorte que la hauteur du contrôle **Label** soit équivalente à une seule ligne de texte.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, la fonctionnalité glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications du code pour le contrôle Label

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

Modifications du code pour rendre une étiquette transparente

L'exemple suivant illustre comment rendre l'arrière-plan d'un contrôle Label transparent. Dans Visual Basic 6.0, s'il existe d'autres contrôles derrière **Label**, ces contrôles apparaissent à travers celle-ci. Dans l'exemple de Visual Basic 2005, tous les contrôles situés derrière **Label** peuvent éventuellement apparaître à travers celle-ci à l'aide des fonctions **BringToFront** et **SendToBack**.

```
' Visual Basic 6.0
Private Sub MakeTransparent()
    Label1.BackStyle = vbTransparent
End Sub
```

VB

```
' Visual Basic 2005
Private Sub MakeTransparent()
  Label1.BackColor = System.Drawing.Color.Transparent
  ' Let controls behind the label show through.
  Label1.SendToBack()
  ' Make the portion of controls behind the label transparent
  Label1.BringToFront()
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle Label

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des alternatives.

Propriétés du contrôle Label

Visual Basic 6.0	Équivalent Visual Basic 2005
Alignment: **0 - Left Justify** **1 - Right Justify**	TextAlign Valeur d'énumération MiddleLeft Valeur d'énumération MiddleRight
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
AutoSize	AutoSize
BackColor	**BackColor** **Remarque** Pour obtenir la liste des constantes, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0. **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
BackStyle	Nouvelle implémentation. Pour plus d'informations, reportez-vous aux différences conceptuelles.
Caption	Text
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Pour obtenir la liste des constantes, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0. **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
LinkItem **LinkMode** **LinkTimeOut** **LinkTopic**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Parent	Méthode FindForm
RightToLeft : **True** **False**	RightToLeft Valeur d'énumération Yes Valeur d'énumération No
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WordWrap	Nouvelle implémentation. **Remarque** Le texte dans le contrôle **Label** de Visual Basic 2005 est renvoyé à la ligne par défaut et ne peut pas être modifié.

Méthodes du contrôle Label

Visual Basic 6.0	**Équivalent Visual Basic 2005**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	Méthode **BringToFront** ou **SendToBack** **BringToFront** **SendToBack**

Événements du contrôle Label

Visual Basic 6.0	Équivalent Visual Basic 2005
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
LinkClose **LinkError** **LinkNotify** **LinkOpen**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **Label** sont mis à niveau vers des contrôles Windows Forms **Label**, et les propriétés, méthodes et événements sont mis à niveau vers leur équivalent. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Concepts

Introduction au contrôle Label

Contrôle Line pour les utilisateurs de Visual Basic 6.0

Le contrôle **Line** dans Visual Basic 6.0 n'a aucun équivalent dans Visual Basic 2005. Toutefois, vous pouvez utiliser les méthodes graphiques pour obtenir les mêmes résultats.

Différences conceptuelles

Dans Visual Basic 6.0, le contrôle **Line** offre un moyen facile de dessiner des lignes sur un formulaire au moment du design. Le contrôle **Line** est un "contrôle léger", c'est-à-dire qu'il n'a pas de handle Windows, également appelé **HWnd**.

Dans Visual Basic 2005, il n'y a aucun équivalent pour le contrôle **Line** et les contrôles légers ne sont plus pris en charge. Toutefois, il existe des méthodes pour dessiner des lignes sur un formulaire au moment du design et au moment de l'exécution.

Au moment du design, vous pouvez dessiner une ligne verticale ou horizontale sur un formulaire en ajoutant un contrôle Label et affectant une chaîne vide à la propriété Text, None à la propriété BorderStyle et la valeur 1 à Width ou Height.

Au moment de l'exécution, vous pouvez dessiner des lignes verticales, horizontales ou diagonales sur le gestionnaire d'événements Paint d'un formulaire en créant un nouvel objet Graphics et appelant ses méthodes.

Dans Visual Basic 6.0, vous pouvez utiliser un contrôle **Line** pour dessiner une ligne sur un contrôle conteneur tel qu'un contrôle **PictureBox** ou **Frame** en ajoutant un contrôle **Line** au conteneur.

Dans Visual Basic 2005, vous pouvez obtenir le même effet en appelant la méthode DrawLine dans l'événement **Paint** du contrôle conteneur.

Modifications du code pour le contrôle Line

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Dessin de lignes horizontales ou verticales

Le code suivant illustre comment dessiner des lignes horizontales et verticales sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, le contrôle **Line** est utilisé ; il suppose que deux contrôles **Line** ont été ajoutés au moment du design. L'exemple Visual Basic 2005 illustre deux méthodes : à l'aide d'un contrôle **Label** et à l'aide de méthodes **Graphics**.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspondait à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Load()
    ' Draw a horizontal line 200 twips from the top of the form.
    Line1.X1 = 0
    Line1.X2 = Me.Width
    Line1.Y1 = 200
    Line1.Y2 = 200
    Line1.BorderColor = vbRed
    Line1.BorderWidth = 1
    ' Draw a vertical line 200 twips from the left of the form.
    Line1.Y1 = 0
    Line1.Y2 = Me.Height
    Line1.X1 = 200
    Line1.X2 = 200
    Line1.BorderColor = vbBlue
    Line1.BorderWidth = 1
```

VB

```
' Visual Basic 2005
' Using Label controls.
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    Dim Line1 As New System.Windows.Forms.Label
    Dim Line2 As New System.Windows.Forms.Label
    ' Draw a horizontal line 14 pixels from the top of the form.
```

```
    Line1.Location = New System.Drawing.Point(0, 14)
    Line1.Size = New System.Drawing.Size(Me.Width, 1)
    Line1.BorderStyle = BorderStyle.None
    Line1.BackColor = System.Drawing.Color.Red
    Line1.Text = ""
    Controls.Add(Line1)
    ' Draw a vertical line 14 pixels from the left of the form.
    Line2.Location = New System.Drawing.Point(14, 0)
    Line2.Size = New System.Drawing.Size(1, Me.Height)
    Line2.BorderStyle = BorderStyle.None
    Line2.BackColor = System.Drawing.Color.Blue
    Line2.Text = ""
    Controls.Add(Line2)
End Sub
```

VB

```
' Visual Basic 2005
' Using Graphics methods
Private Sub Form1Paint(ByVal sender As Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a horizontal line 28 pixels from the top of the form.
    e.Graphics.DrawLine(Pens.Red, 0, 28, Me.Width, 28)
    ' Draw a vertical line 28 pixels from the left of the form.
    e.Graphics.DrawLine(Pens.Blue, 28, 0, 28, Me.Height)
End Sub
```

Dessin d'une ligne diagonale

Le code suivant illustre le dessin d'une ligne diagonale sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, le contrôle **Line** est utilisé ; il suppose qu'un contrôle **Line** a été ajouté au moment du design. L'exemple Visual Basic 2005 utilise les méthodes **Graphics**.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspondait à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Load()
    ' Draw a diagonal line from the top left to the lower right.
    Line1.X1 = 0
    Line1.X2 = Me.ScaleWidth
    Line1.Y1 = 0
    Line1.Y2 = Me.ScaleHeight
    Line1.BorderColor = vbBlack
    Line1.BorderWidth = 1
End Sub
```

VB

```
' Visual Basic 2005
Private Sub FormPaint(ByVal sender As Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a diagonal line from the top left to the lower right.
    e.Graphics.DrawLine(Pens.Black, 0, 0, Me.ClientSize.Width, _
    Me.ClientSize.Height)
End Sub
```

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, les contrôles **Line** verticaux ou horizontaux sont remplacés par les contrôles **Label** Windows Forms, avec la propriété Text ayant la valeur d'une chaîne vide, les propriétés

BorderStyle à **None**, et le BackColor, **Width** et **Height** ayant la valeur correspondant au contrôle d'origine.

Les contrôles **Line** qui n'étaient ni verticaux ni horizontaux ne sont pas mis à niveau. Vous pouvez remplacer le contrôle **Line** au moyen des fonctions **Graphics** intégrées au .NET Framework.

Voir aussi

Concepts

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Contrôle ListBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **ListBox** dans Visual Basic 6.0 est remplacé par le contrôle ListBox ou le contrôle CheckedListBox dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, leur comportement est également différent.

Différences conceptuelles

ListBox activé

Dans Visual Basic 6.0, la propriété **Style** d'un contrôle **ListBox** détermine si une case à cocher est affichée à côté de chaque élément de texte. Plusieurs éléments de **ListBox** peuvent être sélectionnés en sélectionnant la case à cocher à côté de ces éléments même si la propriété **MultiSelect** a la valeur **False**. Il n'existe aucun moyen de déterminer par programme l'état activé d'un élément. Si vous avez requis cette fonction, vous devez utiliser à la place un contrôle **ListView**.

Dans Visual Basic 2005, le nouveau contrôle **CheckedListBox** affiche des cases à cocher à côté de chaque élément ; vous ne pouvez plus afficher des cases à cocher dans le contrôle **ListBox**. Le contrôle **CheckedListBox** vous permet de déterminer par programme l'état activé de chaque élément à travers une collection CheckedItemCollection.

Propriété Columns

Dans Visual Basic 6.0, la propriété **Columns** acceptait un entier spécifiant le nombre de colonnes à afficher.

Dans Visual Basic 2005, la propriété MultiColumn accepte une valeur Boolean et la propriété ColumnWidth un entier spécifiant la largeur en pixels. Si la largeur du contrôle **ListBox** est modifiée au moment de l'exécution, vous devrez peut-être également définir la propriété **ColumnWidth** pour chaque colonne.

Événement ItemCheck

Dans le contrôle **ListBox** de Visual Basic 6.0, lorsqu'un événement **ItemCheck** était déclenché, l'état activé était déjà changé.

Dans le contrôle Visual Basic 2005 **CheckedListBox**, lorsque l'événement ItemCheck est déclenché, l'état d'activation n'a pas encore changé. Vous pouvez obtenir la valeur en attente à partir de l'argument *ItemCheckEventArgs* passé à l'événement.

Modifications apportées au code du contrôle ListBox

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications apportées au code de l'ajout et de la suppression d'éléments dans le contrôle ListBox

Le code suivant illustre comment ajouter et supprimer des éléments **ListBox**.

```
' Visual Basic 6.0
' Add an item at the end of the list.
List1.AddItem "Tokyo"
' Insert an item at the top of the list.
List1.AddItem "Copenhagen", 0
' Remove the first item.
List1.RemoveItem 0
```

VB

```
'Visual Basic 2005
' Add an item at the end of the list.
ListBox1.Items.Add("Tokyo")
' Insert an item at the top of the list.
ListBox1.Items.Insert(0, "Copenhagen")
' Remove the first item.
ListBox1.Items.RemoveAt(0)
```

Modifications apportées au code de l'accès aux éléments spécifiques d'un contrôle ListBox

Le code suivant illustre comment retourner la valeur d'un élément **ListBox**.

```
' Visual Basic 6.0
Private Function GetItemText(i As Integer) As String
```

```
    ' Return the text of the item using the index:
    GetItemText = ListBox1.List(1)
End Function
```

VB

```
' Visual Basic 2005
Private Function GetItemText(ByVal i As Integer) As String
    ' Return the text of the item using the index:
    GetItemText = CStr(ListBox1.Items(i))
End Function
```

Modifications apportées au code de détermination des éléments activés dans un contrôle CheckedListBox

Le code suivant illustre comment déterminer l'état activé d'éléments dans un contrôle **CheckedListBox**.

```
' Visual Basic 6.0
' The Visual Basic 6.0 ListBox control didn't support this scenario,
' a ListView control had to be used instead.
Dim s As String
Dim i As Integer
' Loop through all items
For i = 1 To ListView1.ListItems.Count
    ' If an item is checked, add it to the string
    If ListView1.ListItems(i).Checked = True Then
       s = s & "Checked Item" & CStr(i) & " = " & _
ListView1.ListItems(i) & vbCrLf
    End If
Next
' Determine if any items are checked.
If s <> "" Then
     MsgBox s
End If
```

VB

```
' Visual Basic 2005
' Determine if there are any items checked.
If CheckedListBox1.CheckedItems.Count <> 0 Then
  ' If so, loop through all checked items and print results.
  Dim x As Integer
  Dim s As String = ""
  For x = 0 To CheckedListBox1.CheckedItems.Count - 1
      s = s & "Checked Item " & CStr(x + 1) & " = " & _
      CStr(CheckedListBox1.CheckedItems(x)) & ControlChars.CrLf
  Next x
  MessageBox.Show(s)
End If
```

Équivalences des propriétés, méthodes et événements du contrôle ListBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005

Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Columns	**MultiColumn** et **ColumnWidth**
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.

HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
IntegralHeight	IntegralHeight
ItemData	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de mettre à niveau la propriété ItemData.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
List	Items
ListCount	Count (**List.Count**)
ListIndex	SelectedIndex
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
MultiSelect	SelectionMode
NewIndex	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de mettre à niveau la propriété NewIndex.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
SelCount	Count (**SelectedItem.Count**)
Selected	GetSelected, SetSelected (contrôle **ListBox**) GetItemChecked, SetItemChecked (contrôle **CheckedListBox**)
Style	Propriété de nouvelle implémentation. **0 – Standard** est mappée vers le contrôle **CheckedListBox**, **1 – Checkbox** est mappée vers le contrôle **CheckedListBox**.
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.

Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Nom	Équivalent Visual Basic 2005
AddItem	Add AddRange Insert
Clear	Clear
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
RemoveItem	Remove
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	Méthode BringToFront ou SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005
DblClick	DoubleClick

DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
ItemCheck	**ItemCheck** (**CheckedListBox** uniquement)
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Scroll	Nouvelle implémentation. Pour plus d'informations, consultez Propriété TopIndex et événement Scroll pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lors de la mise à niveau, si la propriété **Style** d'un contrôle **Listbox** de Visual Basic 6.0 prend la valeur **1 – Checkbox**, elle est mise à niveau vers le contrôle **CheckedListBox** ; sinon, elle est mise à niveau vers le contrôle **ListBox**.

Voir aussi

Référence

Vue d'ensemble du contrôle ListBox (Windows Forms)
Vue d'ensemble du contrôle ListBox (Windows Forms)
Vue d'ensemble du contrôle CheckedListBox (Windows Forms)

Contrôle ListView pour les utilisateurs de Visual Basic 6.0

Le contrôle **ListView** dans Visual Basic 6.0 est remplacé par le contrôle ListView dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Propriété DropHighlight

Dans Visual Basic 6.0, lorsque le curseur passe au-dessus d'un objet **ListItem** dans un contrôle **ListView**, la propriété **DropHighlight** est utilisée pour mettre en surbrillance l'objet **ListItem** avec la couleur de surbrillance système. Cette propriété est communément utilisée dans les opérations de glisser-déplacer.

Dans Visual Basic 2005, la propriété **DropHighlight** n'existe plus ; vous pouvez obtenir le même effet par ownerdrawing (également dénommé dessin personnalisé) la surbrillance dans les événements **MouseEnter** et **MouseLeave**. Pour plus d'informations, consultez Contrôles avec prise en charge intégrée des dessins owner-drawn.

Propriété FlatScrollBar

Dans Visual Basic 6.0, la propriété **FlatScrollBar** détermine si la barre de défilement dans un contrôle **ListView** a une apparence bidimensionnelle.

Dans Visual Basic 2005, la propriété **FlatScrollBar** n'existe plus ; les barres de défilement dans le contrôle **ListView** ont toujours une apparence tridimensionnelle.

Propriété HideColumnHeaders

Dans Visual Basic 6.0, la propriété **HideColumnHeaders** détermine si les objets **ColumnHeader** dans un contrôle **ListView** sont masqués dans l'affichage **Report**.

Dans Visual Basic 2005, la propriété **HideColumnHeaders** n'existe plus ; vous pouvez contrôler la visibilité des en-têtes de colonne en définissant la propriété HeaderStyle.

Événement ItemClick

Le contrôle **ListView** de Visual Basic 6.0 possède à la fois un événement **Click** et un événement **ItemClick**. L'événement **ItemClick** utilise un objet **ListItem** comme un argument.

Dans Visual Basic 2005, l'événement **ItemClick** n'est plus pris en charge. À la place, vous pouvez utiliser l'événement **Click** et l'argument **EventSender.FocusedItem** pour déterminer sur quel **ListViewItem** l'utilisateur a cliqué.

Tri

Dans Visual Basic 6.0, le tri d'un contrôle **ListView** est obtenu à l'aide d'une combinaison des propriétés **Sorted**, **SortKey** et **SortOrder**. **Sorted** détermine si le tri est activé, **SortOrder** si le tri est croissant ou décroissant, et **SortKey** spécifie le facteur de tri à utiliser à la place du texte.

Dans Visual Basic 2005, le tri s'effectue à l'aide de la propriété Sorting, qui active le tri et définit l'ordre de tri au cours de la même étape. La propriété **SortKey** est remplacée par la propriété ListViewItemSorter.

Propriété View

Dans Visual Basic 6.0, la propriété **View** détermine comment les éléments sont affichés dans un contrôle **ListView** : comme de grandes icônes avec du texte (**lvwIcon**), comme de petites icônes avec du texte (**lvwSmallIcon**), comme une liste (**lvwList**) ou comme une liste contenant des sous-éléments (**lvwReport**).

Dans Visual Basic 2005, la propriété **View** existe encore, mais les énumérations sont différentes. **lvwIcon** est remplacé par LargeIcon, **lvwSmallIcon** par SmallIcon, **lvwList** par List et **lvwReport** par Details. De plus, il existe un nouvel affichage, Tile, où chaque élément apparaît sous forme d'une icône de taille normale avec l'étiquette de l'élément et les informations de sous-élément à sa droite.

Modifications apportées au code du contrôle ListView

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications apportées au code pour déterminer l'élément sélectionné dans un contrôle ListView

L'exemple de code suivant assigne l'élément **ListView** sélectionné à une variable.

```
' Visual Basic 6.0
Dim theItem As ListItem
theItem = ListView1.SelectedItem
```

VB

```
' Visual Basic 2005
Dim theItem As ListViewItem
If ListView1.SelectedItems.Count > 0 Then
    theItem = ListView1.SelectedItems(0)
Else
    theItem = Nothing
End If
```

Modifications apportées au code pour déterminer tous les éléments sélectionnés dans un contrôle ListView

L'exemple de code suivant affiche un message avec le texte de chaque élément sélectionné dans un contrôle **ListView**.

```
' Visual Basic 6.0
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        MsgBox(ListView1.SelectedItem
    End If
Next i
```

VB

```
' Visual Basic 2005
For Each selectedItem As ListViewItem In ListView1.SelectedItems
    MsgBox(selectedItem.Text)
Next
```

Équivalences des propriétés, méthodes et événements du contrôle ListView

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
Arrange	Alignment
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
ColumnHeaderIcons	ImageIndex, ImageKey

ColumnsHeaders	Columns
Container	Parent
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
DropHighlight	Nouvelle implémentation.
FlatScrollBar	Nouvelle implémentation.
Font Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Icons	LargeImageList
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
ListItems	Items
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.

MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
Picture	BackgroundImage
PictureAlignment	BackgroundImageTiled
SelectedItem	SelectedItems
SmallIcons	SmallImageList
Sorted **SortOrder**	**Sorting**
SortKey	**ListViewItemSorter**
TextBackground	BackColor
ToolTipText	Composant ToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Nom	**Équivalent Visual Basic 2005.**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
FindItem	FindItemWithText

GetFirstVisible	TopItem
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Refresh	RedrawItems
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
StartLabelEdit	BeginEdit
ZOrder	Méthode BringToFront ou SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005.
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
ItemClick	Pas d'équivalent direct.
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, tous les contrôles **ListView** sont mis à niveau vers des contrôles **ListView** Windows Forms. Lorsque les propriétés, méthodes et événements n'ont aucun équivalent ou

présentent des différences de comportement, des notes ou avertissements de mise à niveau sont ajoutés à votre code.

Voir aussi

Autres ressources

ListView, contrôle (Windows Forms)

Contrôle MaskEdBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **MaskEdBox** de Visual Basic 6.0 est remplacé par le contrôle **MaskedTextBox** Windows Forms dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Dans Visual Basic 6.0, la propriété **AutoTab** détermine si le contrôle suivant dans l'ordre de tabulation reçoit le focus dès que la propriété **Text** d'un contrôle **MaskEdBox** est remplie de données valides.

Dans Visual Basic 2005, il n'existe aucun équivalent de la propriété **AutoTab**. Vous pouvez obtenir le même effet en surveillant et en validant la saisie, et en définissant vous-même le focus sur le contrôle suivant.

Dans Visual Basic 6.0, la propriété **PasswordChar** est de type **String** ; dans Visual Basic 2005, elle est de type **Char**.

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, le glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Équivalences des propriétés, méthodes et événements du contrôle MaskedEdBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Visual Basic 2005 Équivalent
AllowPrompt	AllowPromptAsInput
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
AutoTab	Nouvelle implémentation.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
ClipMode	SkipLiterals
ClipText	Text **Remarque** La propriété de Visual Basic 6.0 **ClipText** exclut les littéraux. Dans Visual Basic 2005 les littéraux sont inclus sauf si **SkipLiterals** a la valeur **true**.
Container	Parent

DataBinding **DataChanged** **DataField** **DataFormat**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Format	Nouvelle implémentation. Utilisez des fonctions de mise en forme telles que FormatNumber, fonction (Visual Basic) ou FormatDateTime, fonction (Visual Basic).
FormattedText	**Text** **Remarque** La propriété de Visual Basic 6.0 **ClipText** exclut les littéraux. Dans Visual Basic 2005 les littéraux sont inclus sauf si **SkipLiterals** a la valeur **true**.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.

Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
PromptInclude	TextMaskFormat
SelLength	SelectionLength
SelStart	SelectionStart
SelText	SelectedText
ToolTipText	Composant ToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	Visual Basic 2005 Équivalent

Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	Fonction BringToFront ou SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating
ValidationError	MaskInputRejected

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **MaskEdBox** sont mis à niveau vers des contrôles Windows Forms **MaskedTextBox**, et les propriétés, méthodes et événements sont mis à niveau vers leurs équivalents. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Tâches

Procédure pas à pas : utilisation du contrôle MaskedTextBox

Contrôle MaskedTextBox, exemple

Comment : lier des données au contrôle MaskedTextBox

Autres ressources

MaskedTextBox, contrôle (Windows Forms)

Objet MDIForm pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, un **MDIForm** était un formulaire d'un type particulier qui servait de conteneur à une application MDI (Multiple Document Interface). Dans Visual Basic 2005, tout formulaire dont la propriété IsMdiContainer a la valeur **true** peut agir comme un conteneur MDI.

Différences conceptuelles

Formulaires enfants et formulaires MDI multiples

Dans Visual Basic 6.0, le fait de définir la propriété **MDIChild** d'un formulaire enfant identifie celui-ci comme un enfant du formulaire MDI ; une application MDI ne peut posséder qu'un seul formulaire MDI.

Dans Visual Basic 2005, la propriété MdiParent remplace la propriété **MDIChild** et identifie le formulaire MDI qui contiendra le formulaire enfant ; les applications MDI peuvent avoir plusieurs conteneurs MDI.

Propriétés BackColor et Picture

Dans Visual Basic 6.0, la couleur d'arrière-plan d'un **MDIForm** peut être modifiée en définissant la propriété **BackColor** et une image d'arrière-plan peut être affichée en définissant la propriété **Picture**.

Dans Visual Basic 2005, il n'est pas possible de modifier la couleur d'arrière-plan ou d'afficher une image sur un formulaire MDI. Même si vous pouvez définir les propriétés BackColor et BackgroundImage sur le formulaire, l'affectation de la valeur **true** à la propriété **IsMdiContainer** invalide ces paramètres.

Propriété AutoShowChildren

Dans Visual Basic 6.0, la propriété **AutoShowChildren** d'un formulaire MDI permet d'afficher automatiquement un formulaire MDI enfant dès qu'il est chargé. Les formulaires standard ne présentent pas ce comportement ; l'appel à la méthode **Load** permet de charger, mais n'affiche pas le formulaire.

Dans Visual Basic 2005, vous devez appeler explicitement la méthode Show pour afficher un formulaire enfant MDI.

Contrôles et graphiques sur les formulaires MDI

Dans Visual Basic 6.0, il n'est pas possible d'ajouter la plupart des contrôles à un **MDIForm**. Vous pouvez seulement ajouter des contrôles ancrés tels que le contrôle **Menu**, **StatusBar** ou **ToolBar**. De plus, les méthodes graphiques telles que **Circle** ou **Line** ne peuvent pas être utilisées sur un **MDIForm**.

Dans Visual Basic 2005, vous pouvez ajouter un contrôle à un formulaire MDI ; cependant, le comportement peut ne pas être celui que vous attendez. Un contrôle sur un formulaire dont la propriété **IsMdiContainer** a la valeur **true** navigue sur tout formulaire enfant MDI. Même si vous pouvez appeler des méthodes graphiques dans l'événement Paint, les graphiques n'apparaissent pas au moment de l'exécution.

Modifications du code pour les formulaires MDI

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour afficher des formulaires enfants MDI

Le code suivant illustre comment afficher cinq instances d'un formulaire enfant MDI.

```
' Visual Basic 6.0
Dim F(1 To 4) As New Form1
Private Sub MDIForm_Load()
   Dim i As Integer
   Load Form1
   For i = 1 To 4
      F(i).Caption = "Form" & i + 1
      F(i).Show
   Next I
End Sub
```

VB

```
' Visual Basic 2005
Private Sub MDIForm1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load
```

```
    Dim i As Integer = 0
    For i = 0 To 4
        Dim F As New FormChild
        F.Text = "Form " & CStr(i + 1)
        F.Show()
    Next
End Sub
```

Modifications du code pour réorganiser des fenêtres enfants MDI

Le code suivant illustre comment écrire du code pour modifier la disposition des fenêtres dans une application MDI. Il suppose que vous avez créé des éléments de menu pour chaque option.

```
' Visual Basic 6.0
Private Sub menuCascade_Click()
   Me.Arrange vbCascade
End Sub
Private Sub menuTileHorizontally_Click()
   Me.Arrange vbHorizontal
End Sub
Private Sub menuTileVertically_Click()
   Me.Arrange vbVertical
End Sub
```

VB

```
' Visual Basic 2005
Private Sub CascadeToolStripMenuItem_Click(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles CascadeToolStripMenuItem.Click

  Me.LayoutMdi(MdiLayout.Cascade)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TileVerticleToolStripMenuItem_Click(ByVal sender As Object, _
ByVal e As EventArgs) Handles TileVerticalToolStripMenuItem.Click

  Me.LayoutMdi(MdiLayout.TileVertical)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TileHorizontalToolStripMenuItem_Click(ByVal sender As Object, _
ByVal e As EventArgs) Handles TileHorizontalToolStripMenuItem.Click

  Me.LayoutMdi(MdiLayout.TileHorizontal)
End Sub
```

Équivalences des propriétés et des méthodes du formulaire MDI

Le tableau suivant répertorie les propriétés et les méthodes Visual Basic 6.0 ainsi que leurs équivalents dans Visual Basic 2005. Seules les propriétés et méthodes qui sont uniques à l'objet **MDIForm** sont répertoriées ; pour d'autres propriétés, méthodes et événements, consultez Objet Form pour les utilisateurs de Visual Basic 6.0.

Propriétés et méthodes de l'objet MDIForm

Visual Basic 6.0	Équivalent Visual Basic 2005
Propriété **ActiveForm**	ActiveMdiChild

Méthode **Arrange**	LayoutMdi
Propriété **AutoShowChildren**	Nouvelle implémentation. Pour plus d'informations, consultez Modification du comportement d'affichage dans un formulaire MDI.
Propriété **BackColor**	Nouvelle implémentation. L'arrière-plan d'un formulaire MDI est toujours celui qui est défini par défaut (**Control**).
Propriété **Picture**	Nouvelle implémentation. Les formulaires MDI dans Visual Basic 2005 ne peuvent pas afficher directement d'images.

Mettre à niveau des remarques

Lors de la mise à niveau d'une application MDI de Visual Basic 6.0 vers Visual Basic 2005, tous les objets **MDIForm** sont mis à niveau avec des formulaires normaux et leur propriété **IsMdiContainer** a la valeur **true**.

Dans Visual Basic 2005, la propriété **AutoShowChildren** n'existe plus. Pour émuler le comportement de Visual Basic 6.0, une ligne de code est ajoutée lors de la mise à niveau afin que chaque formulaire enfant soit affiché si **AutoShowChildren** a la valeur **true** dans l'application d'origine.

Voir aussi

Concepts

MDI pour les utilisateurs de Visual Basic 6.0
Objet Form pour les utilisateurs de Visual Basic 6.0
Tâches des formulaires pour les utilisateurs de Visual Basic 6.0

Autres ressources

Applications d'interface multidocument (MDI, Multiple Document Interface)

Objet Menu pour les utilisateurs de Visual Basic 6.0

L'objet **Menu** dans Visual Basic 6.0 est remplacé par le contrôle MenuStrip ou le contrôle ContextMenuStrip dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Dans Visual Basic 6.0, un objet **Menu** est créé à l'aide de l'**Éditeur de menus**. Les objets **Menu** sont liés au formulaire pour lequel ils sont créés, mais ils peuvent être modifiés au moment de l'exécution ou être affichés en tant que menus contextuels. Les menus créés avec l'objet **Menu** ne prennent pas en charge en mode natif l'ombrage, les icônes ou les contrôles incorporés ; seuls les menus Style "en deux dimensions" Windows 98 peuvent être créés.

Dans Visual Basic 2005, l'objet **Menu** est remplacé par les composants **MenuStrip** et **ContextMenuStrip**. Les menus peuvent être créés à l'aide de la modification sur place au moment du design ou dans le code. Les composants **MenuStrip** et **ContextMenuStrip** vous permettent de créer des menus Style modernes dans la barre d'outils Office, avec une prise en charge totale des régions ombragées, des icônes et des contrôles incorporés tels que des listes déroulantes. Pour plus d'informations, consultez Vue d'ensemble du contrôle MenuStrip (Windows Forms).

Affichage des menus contextuels

Dans Visual Basic 6.0, vous pouvez afficher un menu contextuel en appelant la méthode **PopupMenu** et en lui passant tout objet **Menu** de niveau supérieur. Par exemple, si vous avez un menu **Modifier** nommé `mnuEdit` avec des sous-menus **Couper**, **Copier** et **Coller**, vous pouvez afficher un menu contextuel avec les commandes **Couper**, **Copier** et **Coller** en appelant `PopupMenu mnuEdit`.

Dans Visual Basic 2005, les menus contextuels sont affichés à l'aide d'un composant **ContextMenuStrip** séparé. Il n'y a aucun équivalent pour la méthode **PopupMenu**, cependant, vous pouvez toujours partager des commandes entre un **MenuStrip** et un **ContextMenuStrip** en créant **ContextMenuStrip** au moment du design et en partageant des gestionnaires d'événements.

Modifications du code pour l'objet Menu

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour afficher des menus contextuels

Le code suivant illustre comment afficher les commandes **Couper**, **Copier** et **Coller** à partir d'un menu **Modifier** existant dans un menu contextuel.

```
' Visual Basic 6.0
Private Sub mnuCut_Click()
    MsgBox "You selected Cut"
End Sub
Private Sub mnuCopy_Click()
    MsgBox "You selected Copy"
End Sub

Private Sub mnuPaste_Click()
    MsgBox "You selected Paste"
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        PopupMenu mnuEdit
    End If
End Sub
```

VB

```
' Visual Basic 2005
' You must first add a ContextMenuStrip component to the form at design
' time and add Cut, Copy, and Paste menu items named
' CutContextMenuItem, CopyContextMenuItem, and PasteContextMenuItem.

Private Sub CutToolStripMenuItem_Click(ByVal sender As System.Object, _
```

```
  ByVal e As System.EventArgs) Handles CutToolStripMenuItem.Click
      MsgBox("Cut")
  End Sub

  Private Sub CopyToolStripMenuItem_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles CopyToolStripMenuItem.Click
      MsgBox("Copy")
  End Sub

  Private Sub PasteToolStripMenuItem_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles PasteToolStripMenuItem.Click
      MsgBox("Paste")
  End Sub
```

VB

```
  Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As _
  System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown

    If e.Button = Windows.Forms.MouseButtons.Right Then
      Me.ContextMenuStrip = ContextMenuStrip1

      AddHandler CutContextMenuItem.Click, _
        AddressOf CutToolStripMenuItem_Click

      AddHandler CopyContextMenuItem.Click, _
        AddressOf CopyToolStripMenuItem_Click

      AddHandler PasteContextMenuItem.Click, _
        AddressOf PasteToolStripMenuItem_Click
    End If
  End Sub
```

Équivalences des propriétés et des méthodes de menu

Les tableaux suivants répertorient les propriétés et les méthodes de Visual Basic 6.0, ainsi que leurs équivalents Visual Basic 2005. Les propriétés et les méthodes, qui ont des noms et des comportements identiques, ne sont pas répertoriées. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Caption	Text (**MenuStripItem**)
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Index	Nouvelle implémentation. Consultez la propriété **Index** de la classe ToolStripItemCollection.
NegotiatePosition	Pas d'équivalent. Cette propriété n'était utilisée que pour la liaison et l'incorporation d'objets (OLE), qui ne sont plus prises en charge.
Parent	Méthode FindForm
WindowList	MdiWindowListItem

Méthodes

Visual Basic 6.0	Équivalent Visual Basic 2005

PopupMenu	Nouvelle implémentation. Utilisez un composant **ContextMenuStrip**.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les appels à la méthode **PopupMenu** ne sont pas mis à niveau ; vous devez créer des menus contextuels pour les remplacer.

Voir aussi

Référence

Vue d'ensemble du contrôle MenuStrip (Windows Forms)

Concepts

Gestion des menus pour les utilisateurs de Visual Basic 6.0

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Contrôle conteneur OLE pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, le contrôle **OLE Container** vous permettait d'ajouter des objets OLE insérables dans vos formulaires. Il n'y a aucun contrôle **OLE Container** dans Visual Basic 2005.

Différences conceptuelles

Si vous avez besoin des fonctionnalités du contrôle **OLE Container**, dans certains cas, vous pourrez probablement être en mesure de substituer le contrôle WebBrowser dans Visual Basic 2005 et de l'utiliser à la place d'un **OLE Container**.

L'une des utilisations principales du contrôle **OLE Container** consistait à incorporer des documents Word ou des feuilles de calcul Excel dans un formulaire. Le contrôle **WebBrowser** encapsule Internet Explorer, qui est capable d'afficher ces types de documents. De plus, vous pouvez utiliser Visual Studio Tools pour Office pour interagir directement avec les applications Office dans Visual Basic 2005.

Voir aussi

Référence

Vue d'ensemble du contrôle WebBrowser

Concepts

Points à considérer avant d'effectuer la mise à niveau

Autres ressources

Visual Studio Tools pour Office

Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0

Le contrôle **OptionButton** de Visual Basic 6.0 est remplacé par le contrôle RadioButton Windows Forms dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Dans Visual Basic 6.0, l'événement **Click** du contrôle **OptionButton** a été déclenché uniquement lorsque sa propriété **Value** a été modifiée en **True**.

Dans Visual Basic 2005, l'événement CheckedChanged du contrôle **RadioButton** est déclenché chaque fois que la valeur de la propriété Checked passe à **True** ou à **False**.

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, la fonctionnalité glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications du code du contrôle OptionButton

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour sélectionner par programme un contrôle OptionButton

Le code suivant montre comment sélectionner un contrôle **OptionButton** ou **RadioButton** parmi deux contrôles au moment de l'exécution.

```
' Visual Basic 6.0
If Option1.Value = True Then
    Option2.Value = True
Else
    Option1.Value = True
End If
```

VB

```
' Visual Basic 2005
If RadioButton1.Checked = True Then
    RadioButton2.Checked = True
Else
    RadioButton1.Checked = True
End If
```

Modifications du code pour déterminer le moment où un contrôle OptionButton est sélectionné

Le code suivant montre comment gérer l'événement **Click** du contrôle **OptionButton** par rapport à l'événement **CheckedChanged** du contrôle **RadioButton**.

```
' Visual Basic 6.0
' The Click event is only fired when the Value is True
Private Sub Option1_Click()
    MsgBox "Option1 was clicked"
End Sub

Private Sub Option2_Click()
    MsgBox "Option2 was clicked"
End Sub
```

VB

```
' Visual Basic 2005
' The CheckChanged event fires each time the RadioButton's Checked
' value changes to either True or False.
```

```
Private Sub RadioButton1_CheckedChanged(ByVal sender As Object, ByVal _
e As System.EventArgs) Handles RadioButton1.CheckedChanged, _
RadioButton2.CheckedChanged
    ' Only execute if the Checked value is True.
    If sender.Checked = True Then
        MsgBox(sender.Name & " was clicked")
    End If
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle OptionButton

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Alignment	TextAlign
Appearance	FlatStyle
BackColor	BackColor **Remarque** Pour obtenir la liste des constantes, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0. **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Caption	Text
Container	Parent
DisabledPicture **DownPicture**	Nouvelle implémentation. Pour plus d'informations, consultez Propriété Style pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Pour obtenir la liste des constantes, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0. **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
MaskColor	Nouvelle implémentation. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Parent	Méthode FindForm
Picture	Image
RightToLeft: **True** **False**	RightToLeft Valeur d'énumération Yes Valeur d'énumération No
Style	Appearance **Remarque** Il existe des différences relatives à la gestion du style Graphical dans Visual Basic 2005. Pour plus d'informations, consultez Propriété Style pour les utilisateurs de Visual Basic 6.0.
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
UseMaskColor	Nouvelle implémentation. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0.
Value	Checked
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	Méthode BringToFront ou SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005
Click	**CheckedChanged** **Remarque** Dans Visual Basic 6.0, l'événement **Click** a été déclenché seulement lorsque son état est passé à **True** ; dans Visual Basic 2005, il est déclenché pour les valeurs **True** et **False**.
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **OptionButton** sont mis à niveau vers des contrôles Windows Forms **RadioButton**, et les propriétés, méthodes et événements sont mis à niveau vers leur équivalent. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Référence

Vue d'ensemble du contrôle RadioButton (Windows Forms)

Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **PictureBox** dans Visual Basic 6.0 est remplacé par le contrôle PictureBox Windows Forms dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

PictureBox comme conteneur

Le contrôle **PictureBox** dans Visual Basic 6.0 est un contrôle conteneur ; en plus d'afficher des images, il peut être utilisé pour grouper et afficher d'autres contrôles.

Le contrôle **PictureBox** dans Visual Basic 2005 n'est pas un contrôle conteneur ; il est utilisé uniquement pour afficher des images. Vous pouvez utiliser le nouveau contrôle Panel en remplacement du contrôle **PictureBox** lorsque vous avez besoin d'un contrôle conteneur.

AutoSize, propriété

Dans Visual Basic 6.0, la propriété **AutoSize** détermine si un contrôle **PictureBox** conserve sa taille au moment du design (`AutoSize = False`) ou est redimensionné dynamiquement pour s'ajuster aux dimensions de l'image (`AutoSize = True`).

Dans Visual Basic 2005, la propriété **AutoSize** est remplacée par la propriété SizeMode ; outre les modes Normal et AutoSize, des options sont également disponibles pour centrer, étirer ou effectuer un zoom sur l'image.

Propriétés et méthodes graphiques

Dans Visual Basic 6.0, plusieurs propriétés et méthodes graphiques permettent de dessiner des lignes, des formes et du texte dans la partie supérieure d'un contrôle **PictureBox**.

Dans Visual Basic 2005, les contrôles n'ont plus de propriétés ou de méthodes graphiques ; vous pouvez encore dessiner dans la partie supérieure d'un contrôle **PictureBox** à l'aide d'un objet **T:System.Drawing.Graphics**. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris les différences relatives à la liaison de données, la gestion des polices, la fonctionnalité glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications du code pour le contrôle PictureBox

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour afficher une image

Le code suivant illustre comment afficher une image dans un contrôle **PictureBox** au moment de l'exécution.

```
' Visual Basic 6.0
Picture1.Picture = LoadPicture(App.Path & "\somepicture.jpg")
```

VB

```
' Visual Basic 2005
PictureBox1.Image = System.Drawing.Bitmap.FromFile( _
  My.Application.Info.DirectoryPath & "\somepicture.jpg")
```

Modifications du code pour effacer un contrôle PictureBox

Le code suivant illustre comment effacer un contrôle **PictureBox** au moment de l'exécution afin qu'aucune image ne soit affichée.

```
' Visual Basic 6.0
Picture1.Picture = LoadPicture("")
```

VB

```
' Visual Basic 2005
If Not (PictureBox1.Image Is Nothing) Then
    PictureBox1.Image.Dispose()
    PictureBox1.Image = Nothing
End If
```

Modifications du code pour dessiner sur un contrôle PictureBox

Le code suivant illustre comment dessiner un cercle rouge de 40 pixels de diamètre au centre d'un contrôle **PictureBox** au moment de l'exécution.

```
' Visual Basic 6.0
Private Sub Picture1_Paint
    Dim x As Integer
    Dim y As Integer
    Picture1.ScaleMode = vbPixels
    x = Picture1.ScaleWidth / 2
    y = Picture1.ScaleHeight / 2
    Picture1.Circle (x, y), 20, vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e _
As System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint
    Dim radius As Integer = 20
    Dim diameter As Integer = radius * 2
    Dim x As Integer = (PictureBox1.Width / 2) - radius
    Dim y As Integer = (PictureBox1.Height / 2) - radius
    e.Graphics.DrawEllipse(Pens.Red, x, y, diameter, diameter)
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle PictureBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Align	Dock
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
AutoRedraw	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
AutoSize	**SizeMode**

BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
BorderStyle	BorderStyle **Remarque** Dans Visual Basic 6.0, la valeur par défaut est **Fixed Single** ; dans Visual Basic 2005, la valeur par défaut est None.
ClipControls	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Container	Parent
CurrentX **CurrentY**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
DrawMode **DrawStyle** **DrawWidth** **FillColor** **FillStyle**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
HasDC **HDC**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Image	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0. **Remarque** Dans Visual Basic 2005, la propriété **Image** n'est pas une propriété graphique. C'est l'équivalent de la propriété **Picture** dans Visual Basic 6.0.
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

LinkItem **LinkMode** **LinkTimeOut** **LinkTopic**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	FindForm
Picture	Image
ScaleHeight **ScaleLeft** **ScaleMode** **ScaleTop** **ScaleWidth**	Nouvelle implémentation. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005**

Circle **Cls**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Line	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
PaintPicture **Point** **Print** **PSet**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Scale **ScaleX** **ScaleY**	Nouvelle implémentation. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
TextHeight **TextWidth**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
ZOrder	Méthode BringToFront ou SendToBack

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005**

Change	Nouvelle implémentation. Il existe des événements (BackgroundImageChanged, SizeChanged, etc.) qui sont déclenchés lorsque des propriétés sont modifiées.
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
KeyDown **KeyPress** **KeyUp**	Nouvelle implémentation. Le contrôle **PictureBox** dans Visual Basic 2005 n'est pas un contrôle conteneur et ne reçoit pas de séquences de touches.
LinkClose **LinkError** **LinkNotify** **LinkOpen**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lors de la mise à niveau, les contrôles **PictureBox** qui contiennent des contrôles sont mis à niveau avec des contrôles **Panel**. Dans Visual Basic 6.0, les contrôles **PictureBox** qui ont été mis à niveau avec les contrôles **Panel**, les images assignées à la propriété **Picture** sont mappées à la propriété **BackgroundPicture** du contrôle **Panel**.

Voir aussi

Référence

Vue d'ensemble du contrôle PictureBox (Windows Forms)

Objet imprimante pour les utilisateurs de Visual Basic 6.0

L'objet **Printer** de Visual Basic 6.0 est remplacé par le composant PrintDocument dans Visual Basic 2005. Son comportement est très différent, mais dans la plupart des cas, la fonctionnalité peut être dupliquée.

Différences conceptuelles

Dans Visual Basic 6.0, l'impression s'effectue en créant un objet **Printer** et en utilisant des méthodes graphiques qui permettent de dessiner du texte et des éléments graphiques sur une page virtuelle. Les propriétés et les méthodes sont utilisées pour définir des attributs d'impression tels que **DeviceName**, **PrintQuality** ou **Copies** ; d'autres propriétés, telles que **Orientation** et **PaperSize** définissent des attributs de la page proprement dite. La méthode **EndDoc** envoie la sortie vers l'imprimante par défaut pour l'application, définie dans la collection **Printers**.

Dans Visual Basic 2005, l'objet **Printer** n'existe plus. À la place, vous utilisez un composant **PrintDocument** pour définir les éléments graphiques et le texte, un objet **PrinterSettings** pour définir des attributs d'imprimante, et une classe **PageSettings** pour définir des attributs de page.

L'impression n'est plus rattachée à un périphérique spécifique, et le concept d'une imprimante par défaut pour une application n'est plus valide. Au lieu de cela, la méthode **PrintPage** du composant **PrintDocument** permet d'imprimer sur n'importe quel périphérique, et l'imprimante par défaut est valide à l'échelle du système. Les composants **PrintDialog**, **PrintPreviewDialog** et **PageSetupDialog** vous permettent de laisser à l'utilisateur le choix de l'imprimante et des options d'impression au moment de l'exécution.

Propriété ColorMode

Dans Visual Basic 6.0, la propriété **ColorMode** spécifie si la sortie s'imprime en monochrome sur une imprimante couleur.

Dans Visual Basic 2005, c'est désormais l'imprimante qui expose cette possibilité sous forme d'une option avancée. La propriété **SupportsColor** de la classe **PrinterSettings** permet de déterminer les capacités couleur d'une imprimante.

Propriété DriverName

Dans Visual Basic 6.0, la propriété **DriverName** de l'objet **Printer** permet de spécifier un pilote d'imprimante. Dans les premières versions de Windows, et dans MS-DOS, les pilotes d'imprimante étaient nécessaires pour traduire une sortie de manière que chaque marque et modèle d'imprimante spécifiques puisse la comprendre. Si l'avènement de Visual Basic 6.0 a rendu cette fonctionnalité complètement superflue, la propriété a été conservée à des fins de compatibilité descendante.

Dans Visual Basic 2005, la propriété **DriverName** n'existe plus ; les pilotes d'imprimante sont gérés par Windows et vous ne pouvez plus spécifier de pilotes distincts.

Propriété HDC

Dans Visual Basic 6.0, la propriété **hDC** de l'objet **Printer** spécifie un handle d'un contexte de périphérique (Device Context) (un lien entre une application Windows, un pilote de périphérique et un périphérique de sortie, tel qu'une imprimante).

Dans Visual Basic 2005, la propriété **hDC** n'existe plus ; une instance d'un composant **PrintDocument** est l'équivalent d'un contexte de périphérique (Device Context).

Remarque
Bien que les handles de contextes de périphérique (Device Context) ne soient plus nécessaires, les objets graphiques dispose nt de méthodes GetHdc et ReleaseHdc qui peuvent s'utiliser dans les scénarios avancés.

Propriété Page

Dans Visual Basic 6.0, la propriété **Page** retourne un compte des pages imprimées depuis le démarrage de votre application ou depuis la dernière utilisation de l'instruction **EndDoc** sur l'objet **Printer**. Cette propriété est souvent utilisée pour numéroter chaque page lors de l'impression.

Dans Visual Basic 2005, les numéros de page ne sont pas suivis ; cependant, vous pouvez facilement assurer ce suivi en définissant une variable dans l'événement **BeginPrint**, puis en l'incrémentant dans l'événement **PrintPage**.

Port, propriété

Dans Visual Basic 6.0, la propriété **Port** retourne le nom du port par lequel un document est envoyé à une imprimante.

Dans Visual Basic 2005, la propriété **Port** n'existe plus ; les contrôles **PrintDialog** et **PrintPreviewDialog** gèrent automatiquement les informations de port.

Propriété RightToLeft

Dans Visual Basic 6.0, la propriété **RightToLeft** détermine la manière dont l'objet **Printer** met en forme la sortie sur une plate-forme bidirectionnelle, telle que la version arabe ou hébraïque de Windows 95.

Dans Visual Basic 2005, la propriété **RightToLeft** n'est plus nécessaire ; la direction de l'impression est contrôlée par les paramètres de localisation dans les versions ultérieures de Windows.

Propriété TrackDefault

Dans Visual Basic 6.0, la propriété **TrackDefault** détermine si un objet **Printer** pointe toujours sur la même imprimante, ou pointe vers une autre imprimante si vous modifiez le paramètre d'imprimante par défaut dans le Panneau de configuration du système d'exploitation. Modifier le paramétrage de la propriété **TrackDefault** en cours d'impression entraîne l'arrêt immédiat de cette tâche.

Dans Visual Basic 2005, la propriété **TrackDefault** n'existe plus ; la propriété **IsDefaultPrinter** de la classe **PrinterSettings** peut être utilisée pour déterminer si une imprimante est ou non l'imprimante par défaut ; cependant, si l'imprimante par défaut est modifiée, l'impression n'est plus interrompue.

Propriété Zoom

Dans Visual Basic 6.0, la propriété **Zoom** détermine le pourcentage d'agrandissement ou de réduction de la sortie imprimée. Prenez par exemple une page de la taille lettre, imprimée avec une propriété **Zoom** de valeur 50. Cette page contient autant de données qu'une page au format 17 x 22 pouces, car le texte et les éléments graphiques imprimés sur la page de taille lettre ont pour échelle la moitié des valeurs de hauteur et largeur d'origine.

Dans Visual Basic 2005, la propriété **Zoom** n'existe plus ; si une imprimante dispose de fonctions de zoom, les paramètres sont exposés automatiquement dans la boîte de dialogue **Impression**. Vous pouvez également utiliser des méthodes graphiques pour mettre la sortie à l'échelle avant de l'affecter à un composant **PrintDocument**.

Propriétés et méthodes graphiques

Dans Visual Basic 6.0, plusieurs propriétés et méthodes graphiques permettent de dessiner des lignes, des formes et du texte dans un objet **Printer**.

Dans Visual Basic 2005, la plupart des objets ne disposent plus de propriétés ni de méthodes graphiques propres ; vous pouvez cependant encore dessiner des lignes, des formes et du texte, en créant et en utilisant un objet **Graphics**. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

Modifications apportées au code de l'objet Printer

En raison des différences considérables entre les modèles d'impression de Visual Basic 6.0 et de Visual Basic 2005, la comparaison des techniques de codage n'a que peu d'intérêt. Pour obtenir un exemple de l'impression dans Visual Basic 2005, consultez Impression, exemple.

Équivalences des propriétés et des méthodes de l'objet Printer

Les tableaux suivants répertorient les propriétés et les méthodes de Visual Basic 6.0, ainsi que leurs équivalents Visual Basic 2005. Les propriétés et les méthodes, qui ont des noms et des comportements identiques, ne sont pas répertoriées. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
ColorMode	Nouvelle implémentation. La propriété SupportsColor de la classe PrinterSettings permet de déterminer les capacités couleur d'une imprimante.
Copies	Propriété Copies de la classe **PrinterSettings**
CurrentX **CurrentY**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

DeviceName	Propriété PrinterName de la classe **PrinterSettings**
DrawMode **DrawStyle** **DrawWidth**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
DriverName	Nouvelle implémentation. Plus nécessaire ; les pilotes d'imprimante sont gérés par Windows.
Duplex	Propriété Duplex de la classe **PrinterSettings**
FillColor **FillStyle**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Font **FontBold** **FontCount** **FontItalic** **FontName** **Fonts** **FontSize** **FontStrikeThru** **FontTransparent** **FontUnderline**	Nouvelle implémentation. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
ForeColor	Nouvelle implémentation. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
hDC	Plus nécessaire. Une instance d'un composant **PrintDocument** est l'équivalent d'un contexte de périphérique (Device Context).
Height	Propriété PaperSize de la classe PageSettings
Orientation	Propriété Landscape de la classe **PageSettings**

Page	Nouvelle implémentation. Le numéro de page actuel n'est pas suivi ; cependant, vous pouvez facilement en assurer l e suivi en définissant une variable dans l'événement BeginPrint et en l'incrémentant dans l'événement PrintPage.
PaperBi n	Propriété PaperSources de la classe **PrinterSettings**
PaperSi ze	Propriété **PaperSize** de la classe **PageSettings**
Port	Plus nécessaire. Le contrôle PrintPreviewDialog définit automatiquement les informations de port.
PrintQu ality	Propriété PrinterResolutions de la classe **PrinterSettings**
RightTo Left	Plus nécessaire. La direction de l'impression est contrôlée par les paramètres de localisation de Windows.
ScaleHe ight **ScaleLe ft** **ScaleM ode** **ScaleTo p** **ScaleWi dth**	Nouvelle implémentation. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
TrackDe fault	Pas d'équivalent direct. La propriété IsDefaultPrinter de la classe **PrinterSettings** permet de déterminer si une impri mante est l'imprimante par défaut.
TwipsP erPixel X **TwipsP erPixel Y**	Plus nécessaire. Dans Visual Basic 2005, les cotes sont toujours exprimées en pixels.
Width	Propriété **PaperSize** de la classe **PageSettings**
Zoom	Plus nécessaire. Si l'imprimante dispose de fonctionnalités de zoom, les paramètres sont automatiquement exposés dans la boîte de dialogue **Imprimer**.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005**
Circle	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
EndDoc	Print
KillDoc	Cancel

Line	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
NewPage	HasMorePages
PaintPicture **PSet**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.
Scale **ScaleX** **ScaleY**	Nouvelle implémentation. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
TextHeight **TextWidth**	Nouvelle implémentation. Pour plus d'informations, consultez Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, aucune instance de l'objet **Printer** n'est mise à niveau et une erreur de mise à niveau est générée. Vous devez supprimer le code pour l'objet **Printer** et ré-implémenter l'impression à l'aide d'un composant **PrintDocument**.

Voir aussi

Tâches

Impression, exemple

Référence

PageSettings Class

PrinterSettings Class

Concepts

Modifications de l'impression pour les utilisateurs de Visual Basic 6.0

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Autres ressources

Prise en charge de l'impression dans les Windows Forms

Collection d'imprimantes pour les utilisateurs de Visual Basic 6.0

La collection Visual Basic 6.0 **Printers** n'a aucun équivalent direct dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, la collection **Printers** est utilisée pour retourner des informations relatives aux imprimantes disponibles sur un système. En général, vous parcourez la collection **Printers** pour rechercher une imprimante dotée de certains attributs, puis définissez cette imprimante comme imprimante par défaut pour votre application. Par exemple, vous pouvez choisir une imprimante laser au lieu d'une imprimante matricielle.

Dans Visual Basic 2005, il n'existe aucun équivalent direct pour la collection **Printers** et le modèle a changé : plutôt que de choisir une imprimante pour votre application, vous devriez laisser vos utilisateurs faire leurs propres choix.

Un composant PrintDialog peut être utilisé pour présenter une liste d'imprimantes disponibles à l'utilisateur. Les propriétés de l'imprimante sélectionnée par l'utilisateur peuvent être récupérées à l'aide de la classe PrinterSettings.

La collection **Printers** de Visual Basic 6.0 a deux propriétés : **Item** et **Count**. Le composant **PrintDialog** n'a pas de propriétés équivalentes. La classe **PrinterSettings** a une collection **InstalledPrinters** comportant **Item** et **Count**. Toutefois, il s'agit d'une collection **String** qui ne peut pas être utilisée pour interroger les attributs d'une imprimante.

Modifications du code de la collection Printers

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

Modifications du code pour retourner une liste d'imprimantes disponibles

L'exemple suivant illustre le remplissage d'un contrôle **ListBox** au moyen d'une liste d'imprimantes disponibles.

```
' Visual Basic 6.0
Dim X As Printer
For Each X In Printers
    List1.AddItem X.DeviceName
Next
```

VB

```
' Visual Basic 2005
Dim i As Integer
Dim pkInstalledPrinters As String
For i = 0 To System.Drawing.Printing.PrinterSettings. _
  InstalledPrinters.Count - 1

  pkInstalledPrinters = System.Drawing.Printing.PrinterSettings. _
    InstalledPrinters.Item(i)
  ListBox1.Items.Add(pkInstalledPrinters)
Next
```

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, aucune instance de la collection **Printers** n'est mise à niveau, et une erreur de mise à niveau est générée. Vous devrez supprimer le code pour la collection **Printers** et modifier votre code de sorte qu'il utilise un composant **PrintDialog**.

Voir aussi

Référence

Vue d'ensemble du composant PrintDialog (Windows Forms)

PrinterSettings Class

Concepts

Modifications de l'impression pour les utilisateurs de Visual Basic 6.0

Autres ressources

Prise en charge de l'impression dans les Windows Forms

Contrôle ProgressBar pour les utilisateurs de Visual Basic 6.0

Le contrôle **ProgressBar** de Visual Basic 6.0 est remplacé par le contrôle ProgressBar Windows Forms dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Propriété Negotiate

Dans Visual Basic 6.0, la propriété **Negotiate** d'un contrôle **ProgressBar** peut être utilisée pour masquer automatiquement le contrôle lorsqu'une barre d'outils est affichée.

Le contrôle **ProgressBar** de Visual Basic 2005 n'a pas de propriété **Negotiate**. Vous pouvez obtenir le même effet en attribuant à la propriété **Visible** la valeur **false** au moment de l'exécution.

Propriété Orientation

Dans Visual Basic 6.0, un contrôle **ProgressBar** peut être affiché horizontalement ou verticalement à l'aide de la propriété **Orientation**. Le contrôle **ProgressBar** de Visual Basic 2005 peut uniquement être affiché horizontalement.

Équivalences des propriétés, méthodes et événements du contrôle ProgressBar

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes noms et comportements ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Align	Dock
Appearance **BorderStyle**	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
Container	Parent
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Height	Height **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HWnd	**Handle**
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.

Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Max	Maximum
Min	Minimum
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
Negotiate	Nouvelle implémentation.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Orientation	Nouvelle implémentation.
Parent	FindForm
Scrolling	Style
ToolTipText	Composant ToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Value	Value **Remarque** Dans Visual Basic 6.0, lorsque **Value** est modifié, l'événement **Change** est déclenché. Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché ; utilisez l'événement ValueChanged à la place.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder: **0—vbBringToFront** **1—vbSendToBack**	Fonction BringToFront ou SendToBack **BringToFront** **SendToBack**

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005.
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, tous les contrôles **ProgressBar** sont mis à niveau vers des contrôles **ProgressBar** Windows Forms. Lorsque les propriétés, méthodes et événements n'ont aucun équivalent ou présentent des différences de comportement, des notes ou avertissements de mise à niveau sont ajoutés à votre code.

Voir aussi

Tâches

Comment : définir la valeur affichée par le contrôle ProgressBar Windows Forms

Autres ressources

ProgressBar, contrôle (Windows Forms)

Contrôle de données RDO (Remote Data Objects) pour les utilisateurs de Visual Basic 6.0

Le contrôle de données RDO (Remote Data Objects) de Visual Basic 6.0 n'a aucun équivalent direct dans Visual Basic 2005.

Différences conceptuelles

Le contrôle de données RDO (Remote Data Objects) de Visual Basic 6.0 est utilisé en tant que mécanisme de liaison des contrôles avec une base de données à l'aide d'objets RDO. Dans Visual Basic 2005, l'architecture de liaison de données a changé, et RDO n'est plus pris en charge. Pour plus d'informations, consultez
Accès aux données pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, aucune instance du contrôle de données RDO (Remote Data Objects) n'est mise à niveau et une erreur de mise à niveau est générée. Vous devez soit modifier votre application Visual Basic 6.0 pour utiliser le modèle ADO (ActiveX Data Objects) avant de mettre à niveau, soit modifier tout code de liaison de données présent dans l'application mise à niveau pour utiliser la nouvelle architecture de données.

Voir aussi

Concepts

Points à considérer avant d'effectuer la mise à niveau
Accès aux données pour les utilisateurs de Visual Basic 6.0

Contrôle RichTextBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **RichTextBox** de Visual Basic 6.0 est remplacé par le contrôle Windows Forms **RichTextBox** dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Propriété AutoVerbMenu

Dans Visual Basic 6.0, la propriété **AutoVerbMenu** d'un contrôle **RichTextBox** est utilisée pour afficher un menu contextuel avec un jeu de commandes standard.

Dans Visual Basic 2005, la propriété **AutoVerbMenu** n'existe plus. Pour fournir un menu contextuel pour un contrôle RichTextBox, utilisez un composant ContextMenuStrip. Pour plus d'informations, consultez La propriété AutoVerbMenu n'a pas été mise à niveau.

Propriété DisableNoScroll

Dans Visual Basic 6.0, la propriété **DisableNoScroll** détermine si les barres de défilement apparaissent estompées lorsqu'il n'y a pas suffisamment de lignes pour nécessiter un défilement vertical ou pas suffisamment de caractères pour nécessiter un défilement horizontal dans un contrôle **RichTextBox**.

Dans Visual Basic 2005, la propriété **DisableNoScroll** n'existe plus. Vous pouvez utiliser la propriété **ScrollBars** pour forcer l'affichage des barres de défilement, mais vous ne pouvez pas les faire apparaître comme indisponibles.

Propriété FileName et méthode LoadFile

Dans Visual Basic 6.0, la propriété **FileName** ou la méthode **LoadFile** peut être utilisée pour charger un fichier texte ou .rtf dans un contrôle **RichTextBox**.

Dans Visual Basic 2005, seule la méthode **LoadFile** est disponible. Vous ne pouvez pas assigner de fichier au moment du design.

Méthode SelPrint

Dans Visual Basic 6.0, la méthode **SelPrint** d'un contrôle **RichTextBox** peut être utilisée pour imprimer le contenu du contrôle. Elle utilise un seul argument, **hDC**, qui spécifie le contexte de périphérique (Device Context).

Dans Visual Basic 2005, la méthode **SelPrint** n'existe plus et les contextes de périphérique (Device Context) ne sont plus utilisés pour l'impression. Le contrôle **RichTextBox** ne fournit pas de méthode pour imprimer son contenu. Toutefois, vous pouvez étendre la classe **RichTextBox** pour utiliser le message **EM_FORMATRANGE**. Vous pouvez envoyer ensuite le contenu d'un **RichTextBox** à un périphérique de sortie, tel qu'une imprimante. Pour plus d'informations, consultez La propriété SelPrint n'a pas été mise à niveau.

Propriété SelTabCount

Dans Visual Basic 6.0, la propriété **SelTabCount** détermine le nombre de positions de tabulation dans le ou les paragraphes sélectionnés, ou dans les paragraphes qui suivent le point d'insertion.

Dans Visual Basic 2005, la propriété **SelTabCount** n'existe plus. Elle est remplacée par le paramètre **Length** de la propriété **SelectionTabs** (**SelectionTabs.Length**).

Méthode Span

Dans Visual Basic 6.0, la méthode **Span** d'un contrôle **RichTextBox** peut être utilisée pour sélectionner une plage de texte basée sur un jeu de caractères spécifiés.

Dans Visual Basic 2005, la méthode **Span** n'existe plus. Vous pouvez utiliser la méthode **Find** avec l'argument **characterSet()** pour rechercher un caractère spécifié, puis utiliser la propriété **SelectedText** pour sélectionner la plage.

Méthode Upto

Dans Visual Basic 6.0, la méthode **Upto** d'un contrôle **RichTextBox** peut être utilisée pour déplacer le point d'insertion jusqu'au premier caractère membre d'un jeu de caractères spécifiés.

Dans Visual Basic 2005, la méthode **Upto** n'existe plus. Vous pouvez utiliser la méthode **Find** avec l'argument **characterSet()** pour rechercher un caractère spécifié, puis utiliser la propriété **SelectionStart** pour déplacer le point d'insertion.

Différences supplémentaires

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, les fonctionnalités de glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications apportées au code du contrôle RichTextBox

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour la recherche dans un contrôle RichTextBox

Le code suivant montre comment effectuer une recherche entre le point d'insertion et la fin d'une phrase dans un contrôle **RichTextBox**, et le mettre en surbrillance. L'exemple requiert un contrôle **RichTextBox** avec une ou plusieurs phrases entrées dans la propriété **Text**, et requiert aussi un contrôle **Button**. Étant donné que le code est appelé à partir de l'événement **Click** du bouton, la propriété **HideSelection** du **RichTextBox** doit avoir la valeur **False** faute de quoi la mise en surbrillance ne sera pas visible.

```
' Visual Basic 6.0
' Make sure that HideSelection is set to False.
Private Sub Command1_Click()
    RichTextBox1.Span ".?!", True, True
End Sub
```

VB

```
' Visual Basic 2005
' Make sure that HideSelection is set to False.
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click

    Dim endChars() As Char = New Char() {".", "!", "?"}
    Dim intEnd As Integer
    intEnd = RichTextBox1.Find(endChars, RichTextBox1.SelectionStart)
    RichTextBox1.SelectionLength = intEnd - RichTextBox1.SelectionStart
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle RichTextBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005.
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
AutoVerbMenu	Nouvelle implémentation. Pour plus d'informations, consultez La propriété AutoVerbMenu n'a pas été mise à niveau.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Container	Parent

DataBindings **DataChanged** **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DisableNoScroll	Nouvelle implémentation.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
FileName	Méthode LoadFile
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.

Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
SelAlignment	SelectionAlignment
SelBold	SelectionFont
SelBullet	SelectionBullet
SelCharOffset	SelectionCharOffset
SelColor	SelectionColor
SelFontName	**SelectionFont**
SelFontSize	**SelectionFont**
SelHangingIndent	SelectionHangingIndent
SelIndent	SelectionIndent
SelItalic	**SelectionFont**
SelLength	SelectionLength
SelProtected	SelectionProtected
SelRightIndent	SelectionRightIndent

SelRTF	SelectedRtf
SelStart	SelectionStart
SelStrikethru	**SelectionFont**
SelTabCount	**SelectionTabs.Length**
SelTabs	SelectionTabs
SelText	SelectedText
SelUnderline	**SelectionFont**
TextRTF	Rtf
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GetLineFromChar	GetLineFromCharIndex

LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	Pas d'équivalent. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SelPrint	Nouvelle implémentation. Pour plus d'informations, consultez La propriété SelPrint n'a pas été mise à niveau.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Span	Pas d'équivalent direct. Utilisez la méthode **Find** avec un argument **characterSet**.
Upto	Pas d'équivalent direct. Utilisez la méthode **Find** avec un argument **characterSet**.
ZOrder	Fonction BringToFront ou SendToBack

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

SelChange	SelectionChanged
Validate	Validating

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **RichTextBox** sont mis à niveau vers des contrôles Windows Forms **RichTextBox**, et les propriétés, méthodes et événements sont mis à niveau vers leurs équivalents. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Référence

Vue d'ensemble du contrôle RichTextBox (Windows Forms)

Autres ressources

RichTextBox, contrôle (Windows Forms)

Objet Screen pour les utilisateurs de Visual Basic 6.0

L'objet **Screen** de Visual Basic 6.0 n'a aucun équivalent direct dans Visual Basic 2005, mais la plupart de ses fonctionnalités peuvent être dupliquées au moyen du .NET Framework.

Différences conceptuelles

Dans Visual Basic 6.0, l'objet **Screen** fournit l'accès au formulaire et au contrôle actif dans votre application, fournit des informations sur l'écran où l'application est affichée, et vous permet de contrôler l'apparence du curseur.

Dans Visual Basic 2005, il n'y a aucun équivalent direct de l'objet **Screen**, mais la plupart de ses fonctionnalités peuvent être dupliquées au moyen du .NET Framework.

Remarque

Visual Basic 2005 dispose d'une propriété **Screen** : **My.Computer.Screen**. Contrairement à l'objet Visual Basic 6.0 **Screen**, **My.Computer.Screen** retourne uniquement des informations en lecture seule relatives à l'écran, telles que son nom de péri phérique, sa zone de travail et sa profondeur chromatique. Pour plus d'informations, consultez My.Computer.Screen, propriété

Propriété ActiveControl

Dans Visual Basic 6.0, la propriété **ActiveControl** de l'objet **Screen** sert à déterminer le contrôle qui a le focus. La propriété **ActiveControl** peut être utilisée dans une capacité globale, comme dans `Screen.ActiveControl`, où le contrôle actif est retourné sur le formulaire actuellement sélectionné. Si un formulaire spécifique est référencé, par exemple, `Form2.ActiveControl`, **ActiveControl** spécifie le contrôle qui aura le focus si le formulaire référencé est actif.

Dans Visual Basic 2005, il n'y a plus de propriété globale **ActiveControl** ; chaque instance d'un formulaire dispose de sa propre propriété **ActiveControl**. Lors du référencement d'un formulaire spécifique, cette propriété fonctionne exactement comme dans Visual Basic 6.0. Pour déterminer le contrôle actif sur le formulaire actuellement sélectionné, vous devez d'abord parcourir la collection **OpenForms** et vérifier la propriété **ContainsFocus** pour déterminer quel formulaire est actif.

ActiveForm, propriété

Dans Visual Basic 6.0, la propriété **ActiveForm** de l'objet **Screen** sert à déterminer le contrôle qui a le focus. Si un formulaire MDI parent a le focus, **ActiveForm** retourne le dernier formulaire MDI enfant à avoir eu le focus.

Dans Visual Basic 2005, il n'y a plus de propriété **ActiveForm** globale. Pour déterminer le formulaire actif, vous devez effectuer une itération sur la collection **OpenForms** et rechercher le formulaire dont la propriété **ContainsFocus** a pour valeur **True**.

les formulaires MDI parent de Visual Basic 2005 (tout formulaire dans lequel **IsMDIContainer** a pour valeur **True**) ont une propriété **ActiveMDIChild** qui peut être utilisée pour retourner le formulaire enfant actif sans utiliser la collection **OpenForms**.

Propriété MousePointer

Dans Visual Basic 6.0, la propriété **MousePointer** de l'objet **Screen** est utilisée pour modifier l'apparence du curseur; une fois définie, elle s'applique à tous les formulaires de l'application.

Dans Visual Basic 2005, il n'y a plus une propriété **MousePointer** globale ; chaque formulaire dispose d'une propriété **Cursor** propre, qui peut être utilisée pour modifier l'aspect du curseur uniquement pour ce formulaire.

Propriétés TwipsPerPixel

Dans Visual Basic 6.0, les propriétés **TwipsPerPixelX** et **TwipsPerPixelY** de l'objet **Screen** sont utilisées pour convertir des cotes d'écran d'un twip logique (l'unité de mesure standard dans Visual Basic 6.0.) en pixels.

Dans Visual Basic 2005, les pixels sont l'unité de mesure standard ; il n'est plus nécessaire de convertir.

Modifications du code pour l'objet Screen

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour déterminer le contrôle actif

Le code suivant montre comment copier le texte du contrôle actuellement sélectionné sur le formulaire actuellement sélectionné dans le Presse-papiers.

```
' Visual Basic 6.0
If TypeOf Screen.ActiveControl Is TextBox Then
    Clipboard.SetText Screen.ActiveControl.Text
End If
```

VB

```
' Visual Basic 2005
Dim i As Integer
For i = 0 To My.Application.OpenForms.Count - 1
  If My.Application.OpenForms.Item(i).ContainsFocus Then
    If TypeOf (My.Application.OpenForms.Item(i).ActiveControl) _
      Is TextBox Then
        My.Computer.Clipboard.SetText(My.Application.OpenForms. _
          Item(i).ActiveControl.Text)
    End If
  End If
Next
```

Modifications du code pour déterminer le formulaire actif

Le code suivant illustre la modification de la légende du formulaire en cours de sélection.

```
' Visual Basic 6.0
Screen.ActiveForm.Caption = "This is the selected form"
```

VB

```
' Visual Basic 2005
Dim i As Integer
For i = 0 To My.Application.OpenForms.Count - 1
  If My.Application.OpenForms.Item(i).ContainsFocus Then
    My.Application.OpenForms.Item(i).Text = _
      "This is the selected form"
  End If
Next
```

Modifications du code pour déterminer le formulaire actif dans une application MDI

Le code suivant illustre la modification de la légende du formulaire MDI enfant en cours de sélection.

```
' Visual Basic 6.0
Screen.ActiveForm.Caption = "This is the selected child form"
```

VB

```
' Visual Basic 2005
Me.ActiveMdiChild.Text = "This is the selected child form"
```

Équivalences des propriétés d'objet Screen

Le tableau suivant répertorie les propriétés Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005. Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005

ActiveControl	**My.Application.OpenForms(0).ActiveControl**
ActiveForm	**My.Application.OpenForms(0).ContainsFocus** ou ActiveMdiChild (applications MDI)
FontCount **Fonts**	Nouvelle implémentation. Le comportement d'énumération des polices est différent. Pour plus d'informations, consultez Gestion des polices pour les utilisateurs de Visual Basic 6.0.
Height	**My.Computer.Screen.Bounds.Height**
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	**System.Windows.Forms.Cursor**
TwipsPerPixelX **TwipsPerPixelY**	Nouvelle implémentation. Dans Visual Basic 2005, les coordonnées sont exprimées en pixels et les twips ne sont pas utilisés en tant qu'unité de mesure.
Width	**My.Computer.Screen.Bounds.Width**

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, toutes les propriétés de l'objet **Screen** sont mises à niveau vers leurs équivalents Visual Basic 2005 respectifs. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Référence

My.Application.OpenForms, propriété

My.Computer.Screen, propriété

Concepts

Objet App pour les utilisateurs de Visual Basic 6.0

Gestion des polices pour les utilisateurs de Visual Basic 6.0

Contrôle Shape pour les utilisateurs de Visual Basic 6.0

Le contrôle **Shape** dans Visual Basic 6.0 n'a aucun équivalent dans Visual Basic 2005. Toutefois, vous pouvez utiliser les méthodes Graphics pour obtenir les mêmes résultats.

Différences conceptuelles

Dans Visual Basic 6.0, le contrôle **Shape** offre un moyen facile de dessiner des rectangles, des cercles et d'autres formes sur un formulaire au moment du design. Le contrôle **Shape** est un contrôle "léger" ce qui signifie qu'il n'a pas de handle Windows, également appelé **HWnd**.

Dans Visual Basic 2005, il n'y a aucun équivalent pour le contrôle **Shape** et les contrôles légers ne sont plus pris en charge. Toutefois, il existe des méthodes pour dessiner des formes sur un formulaire au moment du design et au moment de l'exécution.

Au moment du design, vous pouvez dessiner une forme carrée ou rectangulaire sur un formulaire en ajoutant un contrôle Label et en affectant une chaîne vide à la propriété Text, FixedSingle à la propriété BorderStyle et la couleur et les dimensions souhaitées à BackColor, Width et Height.

Au moment de l'exécution, vous pouvez dessiner des rectangles, des ellipses et des formes complexes dans le gestionnaire d'événements Paint d'un formulaire en créant un nouvel objet de la classe **Graphics** et appelant ses méthodes.

Dans Visual Basic 6.0, vous pouvez utiliser un contrôle **Shape** pour dessiner une forme sur un contrôle conteneur tel qu'un contrôle **PictureBox** ou **Frame** en ajoutant un contrôle **Shape** au conteneur.

Dans Visual Basic 2005, vous pouvez obtenir le même effet en appelant les méthodes **Graphics** dans l'événement `Paint` du contrôle conteneur.

Modifications du code pour le contrôle Shape

Les exemples de code suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour dessiner des formes rectangulaires

Le code suivant illustre le dessin d'un rectangle plein sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, le contrôle **Shape** est utilisé ; il suppose qu'un contrôle **Line** a été ajouté au moment du design. L'exemple Visual Basic 2005 illustre deux méthodes différentes : à l'aide d'un contrôle **Label** et à l'aide de méthodes **Graphics**.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspondait à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Load()
    ' Show a solid red rectangle 200 twips from the top left.
    Shape1.Top = 200
    Shape1.Left = 200
    Shape1.FillColor = vbRed
    Shape1.FillColor= vbFSSolid
    Shape1.BorderColor = vbRed
End Sub
```

VB

```
' Visual Basic 2005
' Using a Label control
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    Dim Shape1 As New System.Windows.Forms.Label
    ' Show a solid red rectangle 14 pixels from the top left.
    Shape1.Location = New System.Drawing.Point(14, 14)
    Shape1.Size = New System.Drawing.Size(200, 100)
    Shape1.BorderStyle = BorderStyle.None
    Shape1.BackColor = System.Drawing.Color.Red
    Shape1.Text = ""
```

```
    Controls.Add(Shape1)
End Sub
```

VB

```
' Visual Basic 2005
' Using Graphics methods
Private Sub Form2_Paint(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a solid blue rectangle below the red rectangle.
    e.Graphics.FillRectangle(Brushes.Blue, 14, 120, 200, 100)
End Sub
```

Modifications du code pour dessiner un cercle

Le code suivant montre comment dessiner un cercle sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, le contrôle **Shape** est utilisé ; il suppose qu'un contrôle **Shape** a été ajouté au moment du design. L'exemple Visual Basic 2005 utilise les méthodes **Graphics**.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspondait à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Load()
    Draw a 1000 twip diameter red circle
    Shape1.Top = 0
    Shape1.Left = 0
    Shape1.Height = 1000
    Shape1.Width = 1000
    Shape1.Shape = vbShapeCircle
    Shape1.BorderColor = vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form3_Paint(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a 70 pixel diameter red circle.
    e.Graphics.DrawEllipse(Pens.Red, 0, 0, 70, 70)
End Sub
```

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **Shape** sont remplacés par des contrôles **Label** Windows Forms, avec la propriété **BorderStyle** ayant la valeur **FixedSingle** et les propriétés BackColor, **Width** et **Height** ayant la valeur correspondant au contrôle d'origine.

Remarque

Les contrôles **Label** peuvent être ajoutés au formulaire dans un ordre différent de celui des contrôles **Shape** d'origine. Pour l es formes qui se superposent, il peut être nécessaire d'appeler la fonction BringToFront ou SendToBack pour obtenir la mêm e disposition visuelle.

Les contrôles **Shape** dont la propriété **Shape** a la valeur **2-Oval**ou **3-Circle** ne peuvent pas être mis à niveau et sont remplacés par un contrôle **Label** en tant qu'espace réservé. Vous pouvez remplacer le contrôle **Shape** au moyen des fonctions graphiques intégrées au .NET Framework. Pour plus d'informations, consultez
Éléments graphiques pour les utilisateurs de Visual Basic 6.0.

Voir aussi

Concepts

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Contrôle StatusBar pour les utilisateurs de Visual Basic 6.0

Le contrôle **StatusBar** de Visual Basic 6.0 est remplacé par le contrôle StatusStrip Windows Forms dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Le contrôle **StatusBar** de Visual Basic 6.0 contient une collection d'objets **Panel**, pouvant chacun afficher du texte et des images.

Le contrôle **StatusStrip** de Visual Basic 2005 ressemble plus au contrôle **ToolStrip**. En plus de panneaux (maintenant connus comme l'appellation de contrôles **ToolStripStatusLabel**), les contrôles **StatusStrip** peuvent contenir des barres de progression, des menus déroulants et des boutons partagés. Il existe également un contrôle **StatusBar** Windows Forms, mais il est recommandé d'utiliser à la place **StatusStrip**.

Propriété SimpleText

Dans Visual Basic 6.0, la propriété **SimpleText** d'un contrôle **StatusBar** définit le texte qui est affiché lorsque la propriété **Style** a la valeur **sbrSimple**.

Le contrôle **StatusStrip** de Visual Basic 2005 ne prend pas en charge un style simple, et il n'y a plus de propriété **SimpleText**.

Propriété de style

Dans Visual Basic 6.0, la propriété **Style** détermine si un contrôle **StatusBar** affiche plusieurs panneaux (**sbrNormal**) ou juste un grand panneau (**sbrSimple**). En style simple, le style biseau est déclenché sans bordures.

Dans Visual Basic 2005, la propriété **Style** n'existe plus. Pour émuler le style simple vous pouvez ajouter un objet ToolStripStatusLabel seul dans le contrôle **StatusStrip** et définir sa propriété BorderStyle.

Équivalences des propriétés, méthodes et événements du contrôle StatusBar

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes noms et comportements ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Align	Dock
Container	Parent
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
Height	Height **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HWnd	**Handle**
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Panels	ToolStripItemCollection
Parent	FindForm
ShowTips	Composant ToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
SimpleText	Nouvelle implémentation.
Style	Nouvelle implémentation.
ToolTipText	Composant **ToolTip**. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.

Value	Value **Remarque** Dans Visual Basic 6.0, lorsque la propriété **Value** est modifiée, l'événement **Change** est déclenché. Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché ; utilisez l'événement ValueChanged à la place.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	Fonction BringToFront ou SendToBack **BringToFront** **SendToBack**

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
PanelClick	Click

PanelDblClick	DoubleClick

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, tous les contrôles **StatusBar** sont mis à niveau vers des contrôles **StatusBar** Windows Forms. Lorsque les propriétés, méthodes et événements n'ont aucun équivalent ou présentent des différences de comportement, des notes ou avertissements de mise à niveau sont ajoutés à votre code.

Voir aussi

Tâches

Contrôle StatusStrip, exemple

Référence

StatusStrip

Contrôle TextBox pour les utilisateurs de Visual Basic 6.0

Le contrôle **TextBox** de Visual Basic 6.0 est remplacé par le contrôle **TextBox** Windows Forms dans Visual Basic 2005. Les noms de certaines propriétés, méthodes et constantes, et de certains événements ont changé. En outre, dans certains cas, leur comportement est également différent.

Différences conceptuelles

Dans Visual Basic 6.0, la propriété **MaxLength** détermine le nombre des caractères qui peuvent être entrés dans un contrôle **TextBox**. Si vous essayez d'insérer du texte par programme, le texte sera tronqué à la longueur spécifiée par la propriété **MaxLength**.

Dans Visual Basic 2005, le comportement de la propriété **MaxLength** ne s'applique plus au texte qui est ajouté par programme. Pour dupliquer le comportement de Visual Basic 6.0, vous devez tronquer la chaîne vous-même.

Dans Visual Basic 6.0, la propriété **PasswordChar** est de type **String** ; dans Visual Basic 2005, elle est de type **Char**.

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, les opérations de glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications du code du contrôle TextBox

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour la recherche dans un contrôle TextBox

Le code suivant montre comment rechercher une chaîne dans un contrôle **TextBox** et la mettre en surbrillance.

```
' Visual Basic 6.0
Private Sub Form_Load ()
    Text1.Text = "Two of the peak human experiences"
    Text1.Text = Text1.Text & " are good food and classical music."
End Sub
Private Sub Form_Click ()
    Dim Search, Where
    ' Get search string from user.
    Search = InputBox("Enter text to be found:")
    ' Find string in text.
    Where = InStr(Text1.Text, Search)
    If Where Then
        Text1.SetFocus
        Text1.SelStart = Where - 1
        Text1.SelLength = Len(Search)
    Else
        MsgBox "String not found."
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    TextBox1.Text = "Two of the peak human experiences are "
    TextBox1.Text = TextBox1.Text & "good food and classical music."
End Sub
Private Sub Form1_Click(ByVal sender As Object, ByVal e As _
System.EventArgs) Handles Me.Click
    Dim Search As String
    Dim Where As String
    ' Get search string from user.
    Search = InputBox("Enter text to be found:")
    ' Find string in text.
    Where = InStr(TextBox1.Text, Search)
    If Where Then
```

```
            TextBox1.Focus()
            TextBox1.SelectionStart = Where - 1
            TextBox1.SelectionLength = Len(Search)
        Else
            MsgBox("String not found.")
        End If
    End Sub
```

Modifications du code pour limiter le nombre de caractères dans un contrôle TextBox

Le code suivant montre comment utiliser la propriété **MaxLength** pour spécifier un nombre maximal de caractères.

```
' Visual Basic 6.0
Private Sub Form_Load()
    Text1.MaxLength = 5
End Sub
Private Sub SetText()
    ' Only the first five characters will be displayed.
    Text1.Text = "Hello World"
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load2(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    TextBox1.MaxLength = 5
End Sub
Private Sub SetText()
    ' Truncate the string to equal MaxLength.
    TextBox1.Text = Strings.Left("Hello World", TextBox1.MaxLength)
End Sub
```

Équivalences des propriétés, méthodes et événements du contrôle TextBox

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Alignment	TextAlign
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Container	Parent

DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
ForeColor	ForeColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Gestion des couleurs pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

LinkItem **LinkMode** **LinkTimeOut** **LinkTopic**	Nouvelle implémentation. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
Locked	ReadOnly
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm.
SelLength	SelectionLength
SelStart	SelectionStart
SelText	SelectedText
ToolTipText	Composant ToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	Nouvelle implémentation. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	Fonction BringToFront ou SendToBack

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LinkClose **LinkError** **LinkNotify** **LinkOpen**	Nouvelle implémentation. Pour plus d'informations, consultez Échange dynamique de données pour les utilisateurs de Visual Basic 6.0.
LostFocus	Leave

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **TextBox** sont mis à niveau vers des contrôles Windows Forms **TextBox**, et les propriétés, méthodes et événements sont mis à niveau vers leurs équivalents. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Référence

Vue d'ensemble du contrôle TextBox (Windows Forms)

Contrôle Timer pour les utilisateurs de Visual Basic 6.0

Le contrôle **Timer** dans Visual Basic 6.0 est remplacé par le composant Timer dans Visual Basic 2005. Les noms de certaines propriétés et de certains événements ont changé. Dans certains cas, leur comportement est également différent.

Différences conceptuelles

Le contrôle **Timer** de Visual Basic 6.0 est un véritable contrôle qui est placé sur un formulaire au moment de la conception, mais qui n'est pas visible au moment du design.

Le contrôle Visual Basic 2005 **Timer** est un composant ajouté à la barre d'état au moment du design ; en tant que composant, il n'a aucune propriété **Parent**.

Remarque
Outre le composant **Timer**, le .NET Framework possède également deux classes Timer : Timer et Timer.

Propriété Interval

Dans Visual Basic 6.0, vous pouvez désactiver un contrôle **Timer** en définissant la propriété **Interval** à 0.

Dans Visual Basic 2005, la plage inférieure pour la propriété Interval est 1. Si vous attribuez à l'intervalle la valeur 0, il lève une exception au moment de l'exécution. Le composant Visual Basic 2005 **Timer** utilise la propriété Enabled pour vous permettre de le désactiver ou de l'activer.

Modifications du code du contrôle Timer

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour le démarrage et l'arrêt du contrôle Timer

Le code suivant montre comment activer et désactiver un contrôle **Timer** au moment de l'exécution.

```
' Visual Basic 6.0
Public Function TimerOn(Interval As Integer)
    If Interval > 0 Then
        ' Start the timer.
        Timer1.Interval = Interval
    Else
        ' Stop the timer.
        Timer1.Interval = 0
    End If
End Function
```

VB

```
' Visual Basic 2005
Public Sub TimerOn(ByRef Interval As Short)
    If Interval > 0 Then
        ' Start the timer.
        Timer1.Enabled = True
    Else
        ' Stop the timer
        Timer1.Enabled = False
    End If
End Sub
```

Équivalences des propriétés et des événements du contrôle Timer

Le tableau suivant répertorie les propriétés et les événements de Visual Basic 6.0, ainsi que leurs équivalents Visual Basic 2005. Les propriétés et les événements qui ont des noms et des comportements identiques ne sont pas répertoriées.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés et événements

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Propriété **Index**	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0
Propriété **Interval**	**Interval** **Remarque** Le comportement de la propriété **Interval** a changé. Pour plus d'informations, consultez Changement de comportement de la propriété Timer Interval.
Propriété **Parent**	Nouvelle implémentation. **Timer** est un composant et ne peut pas avoir de parent.
Événement **Timer**	**Tick**

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **Timer** sont mis à niveau vers des composants Windows Forms **Timer**. Si un code qui attribue à la propriété **Interval** la valeur 0 est identifié, un avertissement de mise à niveau est ajouté à votre code.

Voir aussi

Référence

Vue d'ensemble du composant Timer (Windows Forms)

Contrôle ToolBar pour les utilisateurs de Visual Basic 6.0

Le contrôle **ToolBar** dans Visual Basic 6.0 est remplacé par le contrôle ToolStrip dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Le contrôle de Visual Basic 6.0 **ToolBar** est un contrôle ActiveX qui est utilisé pour créer des barres d'outils simples avec des boutons et des zones de liste déroulante.

Vous pouvez utiliser le contrôle **ToolStrip** de Visual Basic 2005 pour créer des barres d'outils personnalisées et des éléments d'interface utilisateur ou créer des barres d'outils et des éléments d'interface utilisateur présentant l'apparence et le comportement de Microsoft Windows XP, Microsoft Office ou Microsoft Internet Explorer. Ces éléments d'interface utilisateur prennent en charge le dépassement de capacité et la réorganisation des éléments au moment de l'exécution. Les contrôles **ToolStrip** offrent une expérience de design qui inclut l'activation et la modification sur place, la disposition personnalisée et le rafting, qui correspond à la capacité des barres d'outils de partager l'espace horizontal ou vertical.

Remarque
Visual Basic 2005 possède également un contrôle ToolBar qui provient des versions antérieures. Toutefois, le contrôle **ToolStrip** est le choix recommandé en raison de ses fonctions avancées.

Propriété AllowCustomize, méthode Customize, événement Change

Dans Visual Basic 6.0, la propriété **AllowCustomize** d'un contrôle **ToolBar** peut être utilisée pour afficher une boîte de dialogue **Personnaliser la barre d'outils** standard, qui permet à l'utilisateur de masquer, d'afficher ou de réorganiser des boutons de barre d'outils. La méthode **Customize** peut être utilisée pour appeler la boîte de dialogue par programme. L'événement **Change** est déclenché lorsque l'utilisateur ferme la boîte de dialogue **Personnaliser la boîte à outils**.

Le contrôle **ToolStrip** de Visual Basic 2005 ne possède aucune propriété ou méthode équivalente. Vous devez créer votre propre boîte de dialogue et l'afficher à l'aide de la méthode **ShowDialog**. Pour plus d'informations, consultez Personnalisation de ToolStrip, exemple.

Propriétés DisabledImageList, HotImageList, ImageList

Dans Visual Basic 6.0, les propriétés **DisabledImageList**, **HotImageList** et **ImageList** d'un contrôle **ToolBar** sont utilisées pour associer le contrôle aux contrôles **ImageList**. Différentes images peuvent être associées pour différents états : normal (**ImageList**), désactivé (**DisabledImageList**) et sélectionné (**HotImageList**).

Le contrôle **ToolStrip** de Visual Basic 2005 ne possède pas de propriétés équivalentes. L'objet **ToolStripButton** a une seule propriété **Image**. Attribuer à la propriété **Enabled** du **ToolStripButton** la valeur **False** restitue automatiquement l'image désactivée. Pour émuler le comportement d'état "sélectionné" de Visual Basic 6.0, vous pouvez soit assigner une image différente, soit modifier la propriété **BackColor** du **ToolStripButton** au moment de l'exécution.

Propriété de style

Dans Visual Basic 6.0, la propriété **Style** d'un contrôle **ToolBar** détermine l'endroit où les images sont affichées dans la relation au texte. Lui attribuer la valeur **tbrTransparent** active également la sélection réactive.

Le contrôle **ToolStrip** de Visual Basic 2005 ne possède pas de propriété équivalente. La relation entre l'image et le texte est contrôlée en définissant la propriété **TextImageAlign** du **ToolStripItem**. Contrairement à Visual Basic 6.0, chaque élément peut avoir un alignement différent. La sélection réactive n'est pas directement prise en charge, mais vous pouvez émuler le comportement en modifiant la propriété **Image** dans les événements **MouseEnter** et **MouseLeave**.

Propriété Wrappable

Dans Visual Basic 6.0, la propriété **Wrappable** d'un contrôle **ToolBar** détermine si les boutons de barre d'outils se positionneront automatiquement en cas de redimensionnement de la fenêtre.

Le contrôle **ToolStrip** de Visual Basic 2005 n'a aucun équivalent direct. À la place, il prend en charge l'utilisation d'un menu de dépassement de capacité déroulant. Lorsque vous ajoutez des éléments **ToolStripItem** qui requièrent plus d'espace que l'espace alloué à **ToolStrip**, vu la taille actuelle du formulaire, un **ToolStripOverflowButton** apparaît automatiquement sur le **ToolStrip**. **ToolStripOverflowButton** apparaît et les éléments pour lesquels le dépassement de capacité est activé sont déplacés dans le menu déroulant de dépassement de capacité. Vous pouvez désactiver ce comportement en attribuant à la propriété **CanOverflow** la valeur **False**.

De plus, vous pouvez spécifier le comportement de dépassement de capacité d'un élément **ToolStripItem** individuel en définissant sa propriété **Overflow**. Pour plus d'informations, consultez Comment : gérer le dépassement de capacité de contrôles ToolStrip dans les Windows Forms.

Méthodes RestoreToolBar, SaveToolBar

Dans Visual Basic 6.0, les méthodes **RestoreToolBar** et **SaveToolBar** sont utilisées pour gérer la configuration **ToolBar**. **SaveToolBar** enregistre la configuration dans un fichier d'initialisation (.ini) ; **RestoreToolBar** le retourne à son état d'origine après avoir été personnalisé.

Le contrôle **ToolStrip** de Visual Basic 2005 ne possède pas de méthodes équivalentes. Vous pouvez utiliser le **Concepteur de paramètres** pour gérer des configurations pour le contrôle **ToolStrip**. Pour plus d'informations, consultez Gestion des paramètres de l'application.

Équivalences des propriétés et des méthodes du contrôle ToolBar

Les tableaux suivants répertorient les propriétés et les méthodes de Visual Basic 6.0, ainsi que leurs équivalents Visual Basic 2005. Les propriétés et les méthodes, qui ont des noms et des comportements identiques, ne sont pas répertoriées. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Align	Dock
AllowCustomize	Nouvelle implémentation. Vous devez fournir votre propre boîte de dialogue.
Appearance **BorderStyle**	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
ButtonHeight	Height, Size (**ToolStripButton**)
Buttons	ToolStripItemCollection
ButtonWidth	**Height**, **Size** (**ToolStripButton**)
Container	Parent
Controls	**ToolStripItemCollection**
DataBindings	Nouvelle implémentation. Pour plus d'informations, consultez Accès aux données pour les utilisateurs de Visual Basic 6.0.
DisabledImageList	Nouvelle implémentation. Utilisez la propriété **Image** de l'objet **ToolStripButton**.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Height	**Height** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID **HelpFile**	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HotImageList	Nouvelle implémentation. Utilisez la propriété **Image** de l'objet **ToolStripButton**.
HWnd	**Handle**
ImageList	Nouvelle implémentation. Utilisez la propriété **Image** de l'objet **ToolStripButton**.
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDropMode	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	FindForm
ShowTips	ShowItemToolTips
Style	Nouvelle implémentation. Utilisez la propriété **TextImageAlign** de **ToolStripItem**.
TextAlignment	TextAlign (**ToolStripItem**)
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Value	Value **Remarque** Dans Visual Basic 6.0, lorsque **Value** est modifié, l'événement **Change** est déclenché. Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché ; utilisez l'événement ValueChanged à la place.

WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Wrappable	CanOverflow, Overflow

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Customize	Nouvelle implémentation. Vous devez fournir votre propre boîte de dialogue.
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
RestoreToolbar **SaveToolbar**	Nouvelle implémentation. Utilisez la fonctionnalité Paramètres de l'application.
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder:	Fonction BringToFront ou SendToBack

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Change	Nouvelle implémentation. La boîte de dialogue **Personnaliser la boîte à outils** n'existe plus.
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, tous les contrôles **ToolBar** sont mis à niveau vers des contrôles **ToolStrip** Windows Forms. Lorsque les propriétés, méthodes et événements n'ont aucun équivalent ou présentent des différences de comportement, des remarques ou avertissements de mise à niveau sont ajoutés à votre code.

Voir aussi

Référence

Vue d'ensemble du contrôle ToolStrip (Windows Forms)

Autres ressources

ToolStrip, contrôle (Windows Forms)

Contrôle TreeView pour les utilisateurs de Visual Basic 6.0

Le contrôle **TreeView** dans Visual Basic 6.0 est remplacé par le contrôle TreeView dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Propriété SingleSel

Dans Visual Basic 6.0, la propriété **SingleSel** d'un contrôle **TreeView** détermine si un nœud se développe pour afficher ses nœuds enfants en cas de sélection. Attribuer à cette propriété la valeur **True** provoque le développement du nœud lorsqu'il est sélectionné et le repli du nœud précédemment sélectionné.

Dans Visual Basic 2005, la propriété **SingleSel** n'existe plus. Par défaut, un nœud ne se développe pas lorsqu'il est sélectionné (la valeur par défaut équivaut à attribuer à la propriété **SingleSel** la valeur **False** dans Visual Basic 6.0). Vous pouvez utiliser l'événement AfterSelect pour déterminer quand un nœud a été sélectionné, puis utiliser la méthode Expand ou Collapse pour contrôler le comportement de développement.

Propriétés Image et SelectedImage

Dans Visual Basic 6.0, il est possible d'assigner une propriété **SelectedImage** à un nœud de contrôle **TreeView** sans assigner une propriété **Image**.

Dans Visual Basic 2005, l'objet **TreeNode** d'un contrôle **TreeView** ne peut pas avoir de propriété **SelectedImage** s'il ne possède pas également de propriété **Image**. Si vous devez émuler le comportement de Visual Basic 6.0, vous pouvez assigner une image vide à la propriété **Image**.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris des différences relatives à la liaison de données, la gestion des polices, les opérations de glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Modifications apportées au code du contrôle TreeView

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modification de code pour le développement d'un nœud sélectionné dans un contrôle TreeView

Le code suivant montre comment développer un nœud **TreeView** lorsqu'un utilisateur le sélectionne.

```
' Visual Basic 6.0
TreeView1.SingleSel = True
```

VB

```
' Visual Basic 2005
Private Sub TreeView1_AfterSelect(ByVal sender As Object, ByVal e As _
System.Windows.Forms.TreeViewEventArgs) Handles TreeView1.AfterSelect

    TreeView1.SelectedNode.Expand()
End Sub
```

Modifications de code pour l'ajout de nœuds dans un contrôle TreeView

Le code suivant montre comment ajouter un nouveau nœud en tant qu'enfant du nœud actuellement sélectionné.

```
' Visual Basic 6.0
Dim nodX As Node
Set nodX = TreeView1.Nodes.Add(Node, tvwChild, , "New Node")
```

VB

```
' Visual Basic 2005
```

```
Dim nodX As TreeNode = New TreeNode("New Node")
TreeView1.SelectedNode.Nodes.Add(nodX)
```

Équivalences des propriétés, méthodes et événements du contrôle TreeView

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005.
Appearance	Nouvelle implémentation. Pour plus d'informations, consultez Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0.
BackColor	BackColor **Remarque** Les couleurs sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.
Container	Parent
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Font Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **Remarque** Les polices sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Objet Font pour les utilisateurs de Visual Basic 6.0.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Indentation	Indent

Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
LineStyle	ShowRootLines
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
OLEDragMode **OLEDropMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Parent	Méthode FindForm
Scroll	Scrollable
SingleSel	Nouvelle implémentation. Utilisez l'événement NodeMouseClick.
Style	Propriété de nouvelle implémentation. **0** ou **Standard** correspond au contrôle CheckedListBox, **1** ou **Checkbox** correspond au contrôle **CheckedListBox**.
ToolTipText	Composant ToolTip Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Nom	**Équivalent Visual Basic 2005.**

Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GetVisibleCount	Propriété VisibleCount.
HitTest	GetNodeAt, GetNodeAt
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
StartLabelEdit	BeginEdit
ZOrder	Méthode BringToFront ou SendToBack

Événements

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Collapse	BeforeCollapse
DblClick	DoubleClick
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Expand	AfterExpand
GotFocus	Enter
LostFocus	Leave
NodeCheck	AfterCheck
NodeClick	**NodeMouseClick**

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, tous les contrôles **TreeView** sont mis à niveau vers des contrôles **TreeView** Windows Forms. Lorsque les propriétés, méthodes et événements n'ont aucun équivalent ou présentent des différences de comportement, des notes ou avertissements de mise à niveau sont ajoutés à votre code.

Voir aussi

Tâches

Comment : ajouter des informations personnalisées à un contrôle TreeView ou ListView (Windows Forms)

Comment : itérer dans tous les nœuds d'un contrôle TreeView Windows Forms

Autres ressources

TreeView, contrôle (Windows Forms)

Objet VBControlExtender pour les utilisateurs de Visual Basic 6.0

L'objet Visual Basic 6.0 **VBControlExtender** n'a aucun équivalent dans Visual Basic 2005 et il n'est plus nécessaire d'utiliser un extendeur de contrôle pour héberger un contrôle ActiveX.

Différences conceptuelles

Dans Visual Basic 6.0, l'objet **VBControlExtender** permet de donner à un contrôle ActiveX l'accès aux propriétés d'un formulaire hôte. L'objet **VBControlExtender** est essentiellement utilisé lors de l'ajout dynamique d'un contrôle à la collection **Controls** à l'aide de la méthode **Add**. L'objet **VBControlExtender** est particulièrement utile dans ce but, car il fournit au développeur un jeu générique de propriétés, d'événements et de méthodes.

Dans Visual Basic 2005, les contrôles ActiveX peuvent être hébergés directement dans un formulaire Windows ; ils ne nécessitent plus d'objet extendeur pour accéder aux propriétés du formulaire hôte. Lorsqu'un contrôle ActiveX est ajouté à un projet, un wrapper d'interopérabilité COM est créé. Il vous permet d'utiliser le contrôle ActiveX comme vous utiliseriez n'importe quel autre contrôle.

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, toutes les instances de l'objet **VBControlExtender** sont ignorées. Un wrapper COM Interop est créé pour chaque contrôle ActiveX ; la propriété, la méthode et les événements sont mappés vers leurs équivalents. En l'absence d'équivalent, des avertissements de mise à niveau sont ajoutés au code.

Voir aussi

Tâches

Comment : ajouter des contrôles ActiveX aux Windows Forms

Référence

System.Windows.Forms Namespace

Concepts

Considérations sur l'hébergement d'un contrôle ActiveX dans un Windows Form

Contrôle VScrollBar pour les utilisateurs de Visual Basic 6.0

Le contrôle **VScrollBar** dans Visual Basic 6.0 a été remplacé par le contrôle VScrollBar dans Visual Basic 2005. Certains noms de propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Change, événement

Dans Visual Basic 6.0, lorsque la propriété **Value** d'un contrôle **VScrollBar** est modifiée, l'événement **Change** est déclenché.

Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché; l'événement ValueChanged est déclenché à la place.

Value, propriété

Dans Visual Basic 6.0, lorsque l'événement **Scroll** ou **Change** d'un contrôle **VScrollBar** est déclenché, la valeur de défilement actuelle est assignée à la propriété **Value**.

Lorsque l'événement Scroll est déclenché dans Visual Basic 2005, la propriété **Value** du contrôle n'est pas encore mise à jour. Si vous devez émuler le comportement de Visual Basic 6.0, vous pouvez créer une fonction d'assistance pour récupérer la valeur. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.

LargeChange, propriété

Dans Visual Basic 6.0, la propriété **LargeChange** détermine la quantité des modifications apportées à la propriété **Value** dans un contrôle de barre de défilement lorsque l'utilisateur clique sur la zone entre la case de défilement et la flèche de défilement.

Dans Visual Basic 2005, la valeur par défaut de la propriété **LargeChange** est 10 ; dans Visual Basic 6.0, elle est égale à 1.

Max, propriété

Dans Visual Basic 6.0, la propriété **Max** détermine le paramètre de propriété **Value** maximal dans un contrôle de barre de défilement.

Dans Visual Basic 2005, la propriété **Max** est remplacée par la propriété Maximum ; la valeur par défaut est 100, dans Visual Basic 6.0, elle était de 32767.

Min, propriété

Dans Visual Basic 6.0, la propriété **Min** détermine le paramètre de propriété **Value** minimal dans un contrôle de barre de défilement. La valeur de la propriété **Min** peut être supérieure à celle de la propriété **Max**.

Dans Visual Basic 2005, la propriété **Min** est remplacée par la propriété Minimum, dont la valeur doit toujours être inférieure à celle de la propriété **Maximum**.

Autres différences

De plus, de nombreuses différences conceptuelles s'appliquent à tous les contrôles, y compris les différences relatives à la liaison de données, la gestion des polices, le glisser-déplacer, la prise en charge de l'aide, etc. Pour plus d'informations, consultez Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0.

Équivalences des propriétés, méthodes et événements du contrôle VScrollBar

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes noms et comportements ne sont pas répertoriés. Lorsque des constantes s'appliquent, elles sont présentées en retrait sous le nom de la propriété ou de la méthode. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Le cas échéant, des liens vous renvoient vers des rubriques expliquant les différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
Container	Parent

DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Height	Height **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	**Handle**
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
LargeChange	LargeChange **Remarque** La valeur par défaut dans Visual Basic 6.0 est 1 ; dans Visual Basic 2005, la valeur par défaut est 10.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Max	Maximum **Remarque** La valeur par défaut dans Visual Basic 6.0 est 32767 ; dans Visual Basic 2005, la valeur par défaut est 100.
Min	Minimum **Remarque** Dans Visual Basic 6.0, **Min** peut avoir une valeur supérieure à **Max** ; dans Visual Basic 2005, cela n'est pas autorisé.
MouseIcon	Nouvelle implémentation. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.
MousePointer	Cursor Pour obtenir la liste des constantes, consultez MousePointer pour les utilisateurs de Visual Basic 6.0.
Parent	FindForm

RightToLeft: **True** **False**	RightToLeft Valeur d'énumération Yes Valeur d'énumération No
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
Value	Value **Remarque** Dans Visual Basic 6.0, lorsque **Value** est modifié, l'événement **Change** est déclenché. Dans Visual Basic 2005, l'événement **Change** n'est pas déclenché ; utilisez l'événement **ValueChanged** à la place.
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	Équivalent Visual Basic 2005
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
SetFocus	Focus
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	Fonction BringToFront ou SendToBack **BringToFront** **SendToBack**

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005

Change	**ValueChanged** **Remarque** Le comportement des événements **Change** et **Scroll** a été modifié dans Visual Basic 2005. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
GotFocus	Enter
LostFocus	Leave
Scroll	**Scroll** **Remarque** Le comportement des événements **Change** et **Scroll** a été modifié dans Visual Basic 2005. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.
Validate	Validating

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tout code dans le gestionnaire d'événements **Scroll** pour les contrôles HScrollBar ou **VScrollBar** est modifié en une procédure en raison d'une différence dans la séquence d'événements. Pour plus d'informations, consultez Un événement a été changé en une procédure dans le code.

Voir aussi

Référence

Vue d'ensemble des contrôles HScrollBar et VScrollBar (Windows Forms)

Contrôle WebBrowser pour les utilisateurs de Visual Basic 6.0

Le contrôle **WebBrowser** de Visual Basic 6.0 est remplacé par le contrôle Windows Forms **WebBrowser** dans Visual Basic 2005. Les noms de certaines propriétés, méthodes, événements et constantes ont changé. En outre, dans certains cas, le comportement est également différent.

Différences conceptuelles

Le contrôle de Visual Basic 6.0 **WebBrowser**, également appelé contrôle Microsoft Internet ou Shdocvw.dll, est un contrôle ActiveX qui vous permet d'héberger Internet Explorer à l'intérieur de votre application.

Le contrôle Visual Basic 2005 **WebBrowser** fournit un wrapper managé destiné au contrôle ActiveX **WebBrowser**. Vous pouvez ainsi afficher des pages Web dans vos applications clientes Windows Forms. Vous pouvez utiliser le contrôle **WebBrowser** pour dupliquer la fonctionnalité de navigation Web d'Internet Explorer dans votre application, ou désactiver la fonctionnalité Internet Explorer par défaut et utiliser le contrôle comme une simple visionneuse de documents HTML. Vous pouvez également utiliser le contrôle pour ajouter des éléments d'interface utilisateur DHTML à votre formulaire et masquer le fait qu'ils sont hébergés dans le contrôle **WebBrowser**. Cette approche vous permet de combiner de façon transparente les contrôles Web et Windows Forms dans une même application.

Objet Internet Explorer

Les Contrôles Microsoft Internet de Visual Basic 6.0 se composent de deux composants : le contrôle **WebBrowser** et un objet **Internet Explorer** qui vous permettent de contrôler une instance d'Internet Explorer via OLE Automation.

Il n'y a pas d'équivalent de l'objet **Internet Explorer** dans Visual Basic 2005, et ses propriétés, méthodes et événements connexes n'existent plus. OLE Automation n'est plus pris en charge. Vous pouvez contrôler une instance externe d'Internet Explorer à l'aide d'un langage de script tel que VBScript.

Propriété du document

Dans Visual Basic 6.0, la propriété **Document** retourne une chaîne qui contient le nom d'un document HTML ; dans Visual Basic 2005, elle retourne le document HTML lui-même.

Méthodes Refresh, Refresh2

La méthode de Visual Basic 6.0 **Refresh2** utilise un paramètre qui spécifie le niveau d'actualisation ; la méthode Visual Basic 2005 **Refresh** possède un paramètre optionnel qui peut être utilisé pour spécifier le niveau d'actualisation.

Équivalences des propriétés, méthodes et événements du contrôle WebBrowser

Les tableaux suivants répertorient les propriétés, méthodes et événements Visual Basic 6.0 et leurs équivalents Visual Basic 2005. Ces propriétés, méthodes et événements qui ont les mêmes nom et comportement ne sont pas répertoriés. Sauf indication contraire, toutes les énumérations Visual Basic 2005 sont mappées vers l'espace de noms System.Windows.Forms.

Ce tableau fournit des liens vers les rubriques qui expliquent des différences de comportement. Lorsqu'il n'existe pas d'équivalent direct dans Visual Basic 2005, des liens vous renvoient vers des rubriques vous proposant des alternatives.

Propriétés

Visual Basic 6.0	Équivalent Visual Basic 2005
AddressBar	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
Application	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
Busy	IsBusy
Container	Parent

Document	Document **Remarque** Dans Visual Basic 6.0, la propriété **Document** retourne une chaîne qui contient le nom d'un document HTML ; dans Visual Basic 2005, elle retourne le document HTML lui-même.
DragIcon **DragMode**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
FullName	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
Height	Height, Size **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
HelpContextID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
HWnd	Handle
Index	Nouvelle implémentation. Pour plus d'informations, consultez Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0.
Left	Left **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
LocationName **LocationURL**	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
MenuBar	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
Offline	IsOffline
Parent	Méthode FindForm
Path	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
RegisterAsBrowser	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
RegisterAsDropTarget	AllowWebBrowserDrop
Resizable	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
Silent	ScriptErrorsSuppressed

StatusBar **StatusText**	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
TheaterMode	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
ToolBar	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
ToolTipText	Composant ToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Top	Top **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.
TopLevelContainer	Nouvelle implémentation. Il s'agissait d'une propriété de l'objet **Internet Explorer**.
Type	DocumentType
WhatsThisHelpID	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
Width	Width, **Size** **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Méthodes

Visual Basic 6.0	**Équivalent Visual Basic 2005**
ClientToWindow	PointToClient
Drag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
ExecWB	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
GetProperty	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
Move	SetBounds **Remarque** Les coordonnées sont gérées différemment dans Visual Basic 2005. Pour plus d'informations, consultez Système de coordonnées pour les utilisateurs de Visual Basic 6.0.

Navigate2	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
OLEDrag	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.
PutProperty	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
QueryStatusWB	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
Quit	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
Refresh **Refresh2**	Refresh **Remarque** La méthode de Visual Basic 6.0 **Refresh2** utilise un paramètre qui spécifie le niveau d'actualisation ; la méthode Visual Basic 2005 **Refresh** possède un paramètre optionnel qui peut être utilisé pour spécifier le niveau d'actualisation.
SetFocus	Focus
ShowBrowserBar	Nouvelle implémentation. Il s'agissait d'une méthode de l'objet **Internet Explorer**.
ShowWhatsThis	Nouvelle implémentation. Pour plus d'informations, consultez Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0.
ZOrder	Fonction BringToFront ou SendToBack

Événements

Visual Basic 6.0	Équivalent Visual Basic 2005
BeforeNavigate **BeforeNavigate2**	Navigating
ClientToHostWindow	Nouvelle implémentation. Il s'agissait d'un événement de l'objet **Internet Explorer**.
CommandStateChanged	CanGoBackChanged CanGoForwardChanged
DblClick	DoubleClick
DocumentComplete **DownloadBegin** **DownloadComplete**	DocumentCompleted
DragDrop **DragOver**	Nouvelle implémentation. Pour plus d'informations, consultez Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0.

GotFocus	Enter
LostFocus	Leave
NavigateComplete **NavigateComplete2**	Navigated
NavigateError	Nouvelle implémentation. Il s'agissait d'un événement de l'objet Internet Explorer.
NewWindow **NewWindow2** **NewWindow3**	NewWindow
OnFullScreen **OnMenuBar** **OnQuit** **OnStatusBar** **OnTheaterMode** **OnToolBar** **OnVisible**	Nouvelle implémentation. Il s'agissait d'un événement de l'objet **Internet Explorer**.
PrintTemplateInstantiation **PrintTemplateTearDown** **PrivacyImpactedStateChange**	Nouvelle implémentation. Il s'agissait d'un événement de l'objet **Internet Explorer**.
ProgressChange	ProgressChanged
PropertyChange	Nouvelle implémentation. Il s'agissait d'un événement de l'objet **Internet Explorer**.
SetSecureLockIcon	EncryptionLevelChanged
StatusTextChange	StatusTextChanged
TitleChange	DocumentTitleChanged
UpdatePageStatus	Nouvelle implémentation. Il s'agissait d'un événement de l'objet **Internet Explorer**.
Validate	Validating

WindowClosing **WindowSetHeight** **WindowSetLeft** **WindowSetResizable** **WindowSetTop** **WindowSetWidth**	Nouvelle implémentation. Il s'agissait d'un événement de l'objet **Internet Explorer**.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les contrôles **WebBrowser** sont mis à niveau vers des contrôles Windows Forms **WebBrowser**, et les propriétés, méthodes et événements sont mis à niveau vers leurs équivalents. Des commentaires de mise à niveau sont insérés dans le code aux endroits où des différences de comportement sont susceptibles de se produire.

Voir aussi

Tâches

Comment : utiliser le contrôle WebBrowser dans le .NET Compact Framework

Référence

Vue d'ensemble du contrôle WebBrowser

Autres ressources

WebBrowser, contrôle (Windows Forms)

Modifications des propriétés, méthodes et événements des contrôles pour les utilisateurs de Visual Basic 6.0

De nombreuses propriétés, méthodes et événements des contrôles et objets Visual Basic 6.0 sont remplacés par des propriétés, méthodes et événements Visual Basic 2005 ayant un comportement différent.

Les rubriques suivantes décrivent les modifications des propriétés, méthodes et événements sélectionnés.

Dans cette section

Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0

Décrit les modifications des propriétés **Appearance** et **BorderStyle** de différents contrôles.

Événement Change du contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Décrit les modifications de comportement de l'événement **Change** du contrôle **ComboBox**.

Système de coordonnées pour les utilisateurs de Visual Basic 6.0

Décrit les modifications apportées aux propriétés **Height** et **Width**, à la méthode **Move** et aux paramètres **ScaleMode**.

Événements de formulaire pour les utilisateurs de Visual Basic 6.0

Décrit les modifications de différents événements de niveau formulaire.

Propriété MaxLength pour les utilisateurs de Visual Basic 6.0

Décrit les modifications de la propriété **MaxLength** du contrôle **TextBox**.

MousePointer pour les utilisateurs de Visual Basic 6.0

Décrit les modifications de la propriété et des constantes **MousePointer**.

Propriété Moveable pour les utilisateurs de Visual Basic 6.0

Décrit les modifications de la propriété **Moveable** d'un formulaire.

Propriété Style pour les utilisateurs de Visual Basic 6.0

Décrit les modifications des propriétés **Style**, **Picture**, **Disabled Picture** et **DownPicture** de différents contrôles.

Propriété TopIndex et événement Scroll pour les utilisateurs de Visual Basic 6.0

Décrit les modifications de la propriété **TopIndex** et de l'événement **Scroll** des contrôles **ComboBox** et **ListBox**.

Rubriques connexes

Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0

Décrit les différences conceptuelles générales lorsque vous travaillez avec des formulaires et des contrôles.

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Compare tous les contrôles et les objets Visual Basic 6.0 avec leurs équivalents Windows Forms.

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Répertorie toutes les constantes de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Décrit les différences de langage Visual Basic entre Visual Basic 6.0 et Visual Basic 2005.

Aide pour les utilisateurs de Visual Basic 6.0

Portail de toutes les rubriques de comparaison entre Visual Basic 6.0 et Visual Basic 2005.

Propriétés Appearance et BorderStyle pour les utilisateurs de Visual Basic 6.0

Les propriétés **Appearance** et **BorderStyle** dans Visual Basic 6.0 sont utilisées pour contrôler l'apparence des contrôles.

Différences conceptuelles

Les contrôles **Frame**, **Image**, **Label**, **PictureBox** et **TextBox** de Visual Basic 6.0 prennent en charge les propriétés **Appearance** et **BorderStyle**. La définition d'une combinaison des deux propriétés fait apparaître le contrôle sans aucune bordure, avec une seule ligne de bordure ou avec une bordure tridimensionnelle.

Dans Visual Basic 2005, la propriété **Appearance** n'existe plus pour ces contrôles. Ses fonctionnalités ont été mixées avec la propriété `BorderStyle` pour définir l'apparence en une seule étape.

Les contrôles **CommandButton**, **ListBox** et **ComboBox** de Visual Basic 6.0 ne prennent en charge que la propriété **Appearance**.

Le contrôle Button de Visual Basic 2005 a une propriété FlatStyle qui est l'équivalent de la propriété **Appearance**. Il est possible de donner une apparence à deux dimensions au contrôle ListBox en affectant la valeur FixedSingle à la propriété BorderStyle. Le contrôle **ComboBox** n'a jamais véritablement pris en charge un aspect à deux dimensions dans Visual Basic 6.0 ; il n'a donc pas d'équivalent.

Remarque

Les formulaires Visual Basic 6.0 ont également une propriété **BorderStyle** qui prend en charge un ensemble différent de val eurs. Elle est remplacée par la propriété FormBorderStyle dans Visual Basic 2005.

Modifications du code pour les propriétés Appearance et BorderStyle

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

Modifications du code pour modifier l'apparence des contrôles

L'exemple suivant illustre la définition des propriétés afin de modifier l'apparence des contrôles au moment de l'exécution.

```
' Visual Basic 6.0
' Give a TextBox a flat look with no border.
Text1.Appearance = 0
Text1.BorderStyle = 0
' Give a TextBox a three-dimensional appearance.
Text1.Appearance = 1
Text1.BorderStyle = 1
' Give a CommandButton a flat look
Command1.Appearance = 0
' Give a ListBox a flat look
List1.Appearance = 0
```

VB

```
' Visual Basic 2005
' Give a TextBox a flat look with a single border.
TextBox1.BorderStyle = BorderStyle.None
' Give a TextBox a three-dimensional appearance.
TextBox2.BorderStyle = BorderStyle.Fixed3D
' Give a Button a flat look
Button1.FlatStyle = FlatStyle.Flat
' Give a ListBox a flat look
ListBox1.BorderStyle = BorderStyle.FixedSingle
```

Équivalences des propriétés Appearance et BorderStyle

Le tableau suivant répertorie les constantes **Appearance** et **BorderStyle** de Visual Basic 6.0 ainsi que leurs énumérations

équivalentes dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
BorderStyle = 0 – None	None
Appearance = 0 – Flat **BorderStyle = 1 – Fixed Single**	**FixedSingle**
Appearance = 1 – 3D **BorderStyle = 1 – Fixed Single**	Fixed3D

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 à l'aide de l'Assistant Mise à niveau, les contrôles **Frame** peuvent être mis à niveau vers les contrôles GroupBox ou Panel de Visual Basic 2005. Les mappages **Appearance** et **BorderStyle** s'appliquent uniquement à ces contrôles **Frame** qui sont mis à niveau vers les contrôles **Panel** ; il n'y a aucune propriété **BorderStyle** pour les contrôles **GroupBox**. Pour plus d'informations, consultez Contrôle Frame pour les utilisateurs de Visual Basic 6.0.

Lors de la mise à niveau des contrôles **ComboBox**, **CommandButton** et **ListBox**, la propriété **Appearance** n'est pas mappée. Si la propriété **Appearance** est définie dans le code, un avertissement de mise à niveau est ajouté.

Voir aussi

Concepts

Contrôle ComboBox pour les utilisateurs de Visual Basic 6.0
Contrôle CommandButton pour les utilisateurs de Visual Basic 6.0
Contrôle Frame pour les utilisateurs de Visual Basic 6.0
Contrôle Label pour les utilisateurs de Visual Basic 6.0
Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0
Contrôle TextBox pour les utilisateurs de Visual Basic 6.0
Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0

Événement Change du contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, l'événement **Change** du contrôle **ComboBox** est déclenché lorsque la partie de texte de ce dernier est modifiée ; il ne l'est pas lorsqu'un élément est sélectionné dans la partie de liste du contrôle. La modification par programme du texte d'un élément dans la liste ne déclenche pas l'événement **Change**.

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, l'événement **Change** du contrôle **ComboBox** est mappé à l'événement TextChanged du contrôle Visual Basic 2005 ComboBox. Le comportement de l'événement **TextChanged** diffère de celui de l'événement **Change** ; cette différence peut produire des résultats imprévus dans votre code.

L'événement **TextChanged** Visual Basic 2005 est déclenché chaque fois que le texte est modifié et quelle qu'en soit la raison. Par exemple :

- Lorsque la partie de texte est modifiée.
- Lorsqu'un élément est sélectionné dans la liste.
- Lorsqu'un élément de liste est modifié par programme.
- Lorsque la méthode Add est appelée.

L'exemple suivant illustre ces différences de comportement.

```
' Visual Basic 6.0
Private Sub Form_Load()
   ' Does not raise the Change event.
   Combo1.AddItem "A"
End Sub
Private Sub Form_Click()
   ' Does not raise the Change event.
   Combo1.List(0) = "B"
End If
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load()
   ' Raises the TextChanged event.
   ComboBox1.Items.Add("A")
End Sub
Private Sub Form1_Click(ByVal sender As System.Object, ByVal _
e As System.EventArgs)
   ' Uses the SetItemString method from the VB6 compatibility library;
   ' there is no equivalent method in Visual Basic 2005.
   ' Raises the TextChanged event.
   Microsoft.VisualBasic.Compatibility.VB6. _
      SetItemString(ComboBox1, ComboBox1.Items.Count, "B")
End Sub
```

Étape suivante

- Définissez un point d'arrêt dans la procédure événementielle **TextChanged**, puis exécutez votre code afin de déterminer à quel endroit l'événement est déclenché. Modifiez votre code en conséquence.

Voir aussi

Concepts

Contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Système de coordonnées pour les utilisateurs de Visual Basic 6.0

Les coordonnées pour les formulaires et les contrôles sont exprimées différemment dans Visual Basic 2005 et dans Visual Basic 6.0 ; les méthodes de dimensionnement et de déplacement des formulaires diffèrent également.

Différences conceptuelles

Dans Visual Basic 6.0, les coordonnées pour les formulaires et les contrôles sont exprimées en twips. Dans Visual Basic 2005, les coordonnées sont exprimées en pixels.

Propriétés Height et Width

Dans Visual Basic 6.0, les propriétés **Height** et **Width** sont utilisées pour modifier la taille d'un formulaire ou d'un contrôle. Visual Basic 2005 fournit une propriété `Size` qui vous permet également de modifier à la fois hauteur et largeur en une seule étape.

Move, méthode

Visual Basic 6.0 utilise à la méthode **Move** pour modifier l'emplacement d'un formulaire ou d'un contrôle au moment de l'exécution. Dans Visual Basic 2005, **Move** est remplacé par la méthode SetBounds et les coordonnées sont exprimées en pixels.

ScaleMode, propriété

Visual Basic 6.0 fournit également une propriété **ScaleMode** que vous pouvez utiliser pour définir un système de coordonnées différent. Visual Basic 2005 ne prend en charge qu'un seul système de coordonnées, en pixels. **ScaleMode** et ses propriétés et méthodes (**Scale**, **ScaleHeight**, **ScaleLeft**, **ScaleTop**, **ScaleWidth**, **ScaleX**, **ScaleY**) connexes ne sont pas prises en charge dans Visual Basic 2005. Pour plus d'informations, consultez La propriété ScaleMode n'est pas prise en charge.

Modifications du code pour les coordonnées

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

Redimensionnement d'un formulaire

L'exemple suivant montre comment modifier la taille d'un formulaire au moment de l'exécution.

```
' Visual Basic 6.0
' Measurements are in twips.
Me.Width = 8000
Me.Height = 6000
```

VB

```
' Visual Basic 2005
' Measurements are in pixels.
Me.Size = New System.Drawing.Size(640, 480)
```

Déplacement d'un contrôle

L'exemple suivant montre comment déplacer un contrôle au moment de l'exécution.

```
' Visual Basic 6.0
' Move and retain original size.
CommandButton2.Move 2000, 1000
' Move and resize to 1200 by 800 twips.
CommandButton1.Move 0, 0, 1200, 800
```

VB

```
' Visual Basic 2005
' Move and retain original size.
Button2.SetBounds(20, 10, 0, 0, BoundsSpecified.X Or BoundsSpecified.Y)
```

```
' Move and resize to 120 by 80 pixels.
Button1.SetBounds(0, 0, 120, 80)
```

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, les propriétés Height et Width des formulaires et des contrôles sont automatiquement converties en pixels.

Si l'application d'origine utilisait la propriété **ScaleMode**, l'outil de mise à niveau suppose que la propriété **ScaleMode** avait été définie au moment du design en twips ; si tel n'est pas le cas, la conversion est incorrecte et doit être corrigée. Pour plus d'informations, consultez La propriété ScaleMode n'est pas prise en charge.

Voir aussi

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Modifications des propriétés, méthodes et événements des contrôles pour les utilisateurs de Visual Basic 6.0

Événements de formulaire pour les utilisateurs de Visual Basic 6.0

Il existe quelques différences dans le comportement des événements de formulaire entre Visual Basic 6.0 et Visual Basic 2005.

Différences conceptuelles

Initialize, événement

Dans Visual Basic 6.0, l'événement **Initialize** est utilisé pour exécuter du code avant le chargement d'un formulaire.

Dans Visual Basic 2005, du code d'initialisation doit être ajouté au constructeur de formulaire (**Sub New()**) après l'appel à **InitializeComponent()**, comme le montre l'exemple suivant :

```
' Visual Basic 6.0
Private Sub Form_Initialize()
    MsgBox("The form is loading")
End Sub
```

VB

```
' Visual Basic .NET
Public Sub New()
    ' This call is required by the Windows Form Designer.
    InitializeComponent()
    ' Add any initialization after the InitializeComponent() call.
    MsgBox("The form is loading")
End Sub
```

Remarque

Une utilisation commune de l'événement **Initialize** consiste à afficher un formulaire de "démarrage" pour présenter des info rmations pendant le chargement du formulaire. Les applications Visual Basic 2005 ont une propriété **SplashScreen** qui per met d'afficher automatiquement un formulaire pendant le démarrage des applications. Pour plus d'informations, consultez Comment : spécifier un écran de démarrage pour une application (Visual Basic).

Terminate, événement

Dans Visual Basic 6.0, l'événement **Terminate** est utilisé pour exécuter du code après le déchargement d'un formulaire.

Dans Visual Basic 2005, l'événement **Terminate** n'est plus pris en charge. Le code d'arrêt doit être exécuté à l'intérieur de la méthode **Dispose**, avant l'appel à **MyBase.Dispose()**.

Remarque

La méthode **Dispose** est automatiquement appelée pour le formulaire principal d'une application ; vous devez l'appeler expli citement pour tout autre formulaire.

L'exemple suivant illustre ces différences.

```
' Visual Basic 6.0
Private Sub Form_Terminate()
    MsgBox "The form was terminated"
End Sub
```

VB

```
' Visual Basic .NET
```

```
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso components IsNot Nothing Then
        MsgBox("The form was terminated")
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub
```

Unload, événement

Dans Visual Basic 6.0, l'événement **Unload** possède un argument Cancel ; dans Visual Basic 2005, il est remplacé par l'événement **Closed** qui ne possède pas d'argument Cancel. Si vous devez annuler un déchargement, utilisez plutôt l'événement **Closing**.

Événements de souris de formulaires MDI

Dans Visual Basic 6.0, les formulaires MDI prend en charge les événements de souris. Dans Visual Basic 2005, les événements **Click**, **MouseDown**, **MouseMove** et **MouseUp** ne sont plus pris en charge pour les formulaires MDI, parce que ceux-ci ne disposent pas d'une zone client pour la réception des événements de souris.

Voir aussi

Référence

Form Constructor

Form.Closing Event

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Propriété MaxLength pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, la propriété **MaxLength** d'un contrôle **TextBox** possède un contrôle absolu sur la longueur d'une chaîne pouvant être entrée ou affichée dans la zone de texte. Même si vous entrez une chaîne plus longue par programme, elle est automatiquement tronquée à la longueur définie dans la propriété **MaxLength**.

Dans Visual Basic 2005, la longueur de la chaîne entrée par programme se substitue automatiquement au paramètre de propriété **MaxLength**.

L'exemple suivant illustre cette différence de comportement :

```
' Visual Basic 6.0
Text1.MaxLength = 5
Text1.Text = "Longer than five"
MsgBox Text1.Text       ' Displays "Longe".
```

VB

```
' Visual Basic 2005
Text1.MaxLength = 5
Text1.Text = "Longer than five"
MsgBox(Text1.Text)       ' Displays "Longer than five".
```

Étape suivante

- Vérifiez le design de votre application pour vous assurer que le comportement souhaité est bien de limiter dans tous les cas la longueur de la chaîne entrée dans la zone de texte à celle qui a été définie pour la propriété **MaxLength**.
    - Si tel est bien le comportement souhaité, recherchez chaque occurrence du code définissant la propriété **Text** et ajoutez, pour tronquer la chaîne, un code similaire à celui de l'exemple suivant.

      VB

      ```
      ' Old code
      Text1.Text = SomeString

      ' Replacement code
      If Len(SomeString) > Text1.MaxLength Then
        SomeString = Microsoft.VisualBasic.Left(SomeString, Text1.MaxLength)
      End If

      Text1.Text = SomeString
      ```

    - Si tel n'est pas le comportement souhaité, ne modifiez pas le code. L'utilisateur est toujours limité par le paramètre **MaxLength**, mais vous pouvez assigner par programme des chaînes dont la longueur est supérieure à celle du paramètre **MaxLength**.

Voir aussi

Concepts

Contrôle TextBox pour les utilisateurs de Visual Basic 6.0

MousePointer pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 2005, la propriété **MousePointer** Visual Basic 6.0 a été remplacée par la propriété Cursor et les noms des constantes **MousePointer** ont été changés.

Différences conceptuelles

Dans Visual Basic 6.0, la propriété **MousePointer** d'un formulaire ou contrôle est utilisée pour modifier l'apparence du curseur lorsqu'il passe sur ce formulaire ou contrôle.

Dans Visual Basic 2005, la propriété **Cursor** fournit la même fonctionnalité que le **MousePointer**.

Pointeurs de souris personnalisés

Dans Visual Basic 6.0, vous pouvez utiliser tout fichier icône (.ico) ou curseur (.cur) comme un **MousePointer** personnalisé.

Dans Visual Basic 2005, seuls les fichiers curseur sont prises en charge ; vous pouvez définir un curseur personnalisé au moment de l'exécution en assignant un fichier curseur à la propriété **Cursor**.

Constante vbIconPointer

Visual Basic 6.0 inclut la constante **vbIconPointer MousePointer** pour modifier le curseur en un symbole d'icône (un petit carré dans un carré). Cette constante est fournie uniquement à des fins d'héritage ; sur les systèmes d'exploitation plus récents, elle n'a aucun effet et le curseur par défaut est affiché. Il n'y a aucun équivalent dans Visual Basic 2005.

Modifications du code pour la propriété MousePointer

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modification de l'apparence du curseur

Le code suivant illustre la façon dont le curseur change d'apparence lorsqu'il passe sur un contrôle de zone de texte au moment de l'exécution, en remplaçant la flèche par défaut par un pointeur en sablier. L'exemple Visual Basic 6.0 définit la propriété **MousePointer** ; dans l'exemple de Visual Basic 2005, la propriété **Cursor** est l'équivalent de **MousePointer**. L'exemple de Visual Basic 2005 utilise également le nouvel événement MouseEnter.

```
' Visual Basic 6.0
Private Sub Text1_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
    Text1.MousePointer = vbHourGlass
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBox1_MouseEnter(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles TextBox1.MouseEnter
    TextBox1.Cursor = System.Windows.Forms.Cursors.WaitCursor
End Sub
```

Affichage d'un curseur personnalisé

Le code suivant illustre l'affichage d'un curseur en forme de main lorsque le curseur passe sur un contrôle de zone de texte. L'exemple Visual Basic 6.0 définit à la fois les propriétés **MouseIcon** et **MousePointer** ; l'exemple Visual Basic 2005 affecte à la propriété **Cursor** une nouvelle instance d'un objet Cursor.

```
' Visual Basic 6.0
Private Sub Text1_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
    Text1.MouseIcon = LoadPicture("C:\Windows\Cursors\hmove.cur")
    Text1.MousePointer = vbHourGlass
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBox2_MouseEnter(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles TextBox2.MouseEnter
    TextBox2.Cursor = New System.Windows.Forms.Cursor _
    ("C:\mypath\mycursor.cur")
End Sub
```

Équivalences de la constante MousePointer

Le tableau suivant répertorie les constantes Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
0 – vbDefault	Default
1 – vbArrow	Arrow
2 – vbCrossHair	Cross
3 – vbIBeam	IBeam
4 – vbIconPointer	Obsolète — remplacé par **Default**
5 – vbSizePointer	SizeAll
6 – vbSizeNESW	SizeNESW
7 – vbSizeNS	SizeNS
8 – vbSizeNWSE	SizeNWSE
9 – vbSizeWE	SizeWE
10 – vbUpArrow	UpArrow
11 – vbHourGlass	WaitCursor
12 – vbNoDrop	No
13 – vbArrowHourGlass	AppStarting
14 – vbArrowQuestion	Help
15 – vbSizeAll	**SizeAll**
99 – vbCustom	Aucun équivalent — pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau à Visual Basic 2005, la propriété **MousePointer** est remplacée par la propriété **Cursor**. Le code qui utilise des constantes **MousePointer** est modifié pour utiliser des énumérations Cursors Visual Basic 2005.

Si l'application Visual Basic 6.0 utilise un **MousePointer** personnalisé, le paramètre de design ou le code **MousePointer**

personnalisé ne sera pas mis à niveau et un avertissement sera émis.

De plus, si l'application Visual Basic 6.0 affecte au **MousePointer** la constante **vbIconPointer** au moment du design ou de l'exécution, il sera remplacé par le curseur par défaut lors de la mise à niveau.

Voir aussi

Référence

Impossible de définir un pointeur de souris personnalisé

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Modifications des propriétés, méthodes et événements des contrôles pour les utilisateurs de Visual Basic 6.0

Propriété Moveable pour les utilisateurs de Visual Basic 6.0

La propriété **Moveable** d'un formulaire Visual Basic 6.0 n'a aucun équivalent dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, la propriété **Moveable** d'un formulaire peut avoir la valeur **False** pour empêcher qu'un utilisateur ne déplace le formulaire au moment de l'exécution. Windows Forms de Visual Basic 2005 n'a pas de propriété équivalente.

Bien que cela soit généralement considéré comme une mauvaise conception d'interface utilisateur, vous pouvez parvenir à un comportement similaire dans Visual Basic 2005 en affectant à la propriété FormBorderStyle la valeur None et à la propriété ControlBox la valeur **False**.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, toutes références au moment du design ou de l'exécution à la propriété **Moveable** ne sont pas mises à niveau et un commentaire d'avertissement est ajouté au code.

Voir aussi

Référence

ControlBox

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Objet MDIForm pour les utilisateurs de Visual Basic 6.0

Propriété Style pour les utilisateurs de Visual Basic 6.0

Les procédures d'affichage des images pour Visual Basic 6.0 **CommandButton**, **CheckBox** et les contrôles **OptionButton** diffèrent de ceux qui sont utilisés en Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, l'affectation de la valeur `1 - Graphical` à la propriété **Style** d'un contrôle **CommandButton**, **CheckBox** ou **OptionButton** active le contrôle pour afficher les images. Les propriétés **Picture**, **DownPicture** et **DisabledPicture** sont utilisées pour assigner des images à afficher en réponse à des changements d'état. Par exemple, lorsqu'un contrôle **CheckBox** est activé, l'image **DownPicture** est affichée ; si le contrôle **CheckBox** est désactivé, l'image **DisabledPicture** est affichée.

Visual Basic 2005 ne prend plus en charge les propriétés **Style**, **Picture**, **DownPicture** et **DisabledPicture**. La propriété **Style** n'est plus nécessaire : l'assignation d'une image à la propriété Image a le même effet que l'affectation de `Graphical` à la propriété **Style**. La propriété **Picture** est remplacée par la propriété **Image**. Les fonctionnalités des propriétés **DownPicture** et **DisabledPicture** peuvent être émulées à l'aide d'un contrôle ImageList qui contient plusieurs images.

Mettre à niveau des remarques

Lorsque vous mettez à niveau une application dont la propriété **Style** a la valeur `1 - Graphical`, la propriété **Appearance** du contrôle mis à niveau a la valeur **Button**, et toute image assignée à la propriété **Picture** au moment du design est assignée à la propriété **Image** du contrôle mis à niveau.

Si les propriétés **DownPicture** ou **DisabledPicture** ont été définies au moment du design ou de l'exécution, vous devrez modifier votre application mise à niveau pour utiliser un contrôle **ImageList**. Pour plus d'informations, consultez Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau.

Voir aussi

Tâches

Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau

Concepts

Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0
Contrôle CommandButton pour les utilisateurs de Visual Basic 6.0
Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0
Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0

Propriété TopIndex et événement Scroll pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 ne prend plus en charge la propriété **TopIndex** pour un contrôle ComboBox ou l'événement **Scroll** pour les contrôles ListBox ou **ComboBox**.

Différences conceptuelles

TopIndex, propriété

Dans Visual Basic 6.0, la propriété **TopIndex** est utilisée pour retourner ou définir une valeur désignant l'élément d'un contrôle **ComboBox** ou **ListBox** qui est affiché en première position de la liste. Cette propriété est couramment utilisée pour faire défiler une liste sans y sélectionner d'élément.

Dans Visual Basic 2005, la propriété **ComboBox** ne prend plus en charge la propriété **TopIndex**. Dans la plupart des cas, cela ne devrait poser aucun problème, dans la mesure où la définition de la propriété **TopIndex** n'a d'effet visible que si la propriété **Style** a la valeur **1 - SimpleCombo**. En l'occurrence, vous pouvez émuler le comportement de **SimpleCombo** de **ComboBox** en utilisant un contrôle **ListBox** et un contrôle TextBox ; le contrôle **ListBox** prend toujours en charge la propriété TopIndex.

Scroll, événement

L'événement **Scroll** dans Visual Basic 6.0 est utilisé conjointement avec la propriété **TopIndex** pour exécuter des actions lors du défilement de la liste. Visual Basic 2005 ne prend pas en charge l'événement **Scroll** ; toutefois, dans la plupart des cas, l'événement SelectedIndexChanged doit être un élément de remplacement approprié.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tout code qui fait référence à la propriété **TopIndex** pour un contrôle **ComboBox** ou tout code dans la procédure événementielle **Scroll** pour un contrôle **ComboBox** ou **ListBox** n'est pas mis à niveau. L'Assistant Mise à jour insère un commentaire d'avertissement dans le code ; celui-ci doit être supprimé pour l'application à compiler.

Voir aussi

Référence

SelectedIndexChanged

SelectedIndexChanged

Concepts

Contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Contrôle ListBox pour les utilisateurs de Visual Basic 6.0

Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0

Si les formulaires et les contrôles de Visual Basic 6.0 vous sont familiers, vous reconnaîtrez beaucoup de ressemblances avec les Windows Forms de Visual Basic 2005. Il convient toutefois de noter quelques domaines qui sont conceptuellement différents. Les rubriques suivantes vous familiariseront avec ces différences.

Dans cette section

Les rubriques suivantes présentent les différences conceptuelles qui existent entre travailler avec des formulaires et des contrôles dans Visual Basic 6.0 et dans Visual Basic 2005.

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0

Modifications des boîtes de dialogue pour les utilisateurs de Visual Basic 6.0

Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0

Échange dynamique de données pour les utilisateurs de Visual Basic 6.0

Gestion des polices pour les utilisateurs de Visual Basic 6.0

Positionnement de formulaire pour les utilisateurs de Visual Basic 6.0

Tâches des formulaires pour les utilisateurs de Visual Basic 6.0

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0

Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application

Gestion des menus pour les utilisateurs de Visual Basic 6.0

MDI pour les utilisateurs de Visual Basic 6.0

Modifications de l'impression pour les utilisateurs de Visual Basic 6.0

Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0

Numérotation des versions pour les utilisateurs de Visual Basic 6.0

Rubriques connexes

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Décrit des différences entre les contrôles Visual Basic 6.0 et Windows Forms.

Vue d'ensemble des Windows Forms

Présente les concepts fondamentaux des formulaires.

Nouveautés dans les Windows Forms pour les utilisateurs de Visual Basic 6.0

Décrit les nouvelles fonctionnalités des Windows Forms.

Espaces de noms d'applications Windows dans Visual Studio

Présente les espaces de noms liés aux Windows Forms.

Aide pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers d'autres rubriques décrivant les différences entre Visual Basic 6.0 et Visual Basic 2005.

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

Cette rubrique compare la gestion des couleurs dans Visual Basic 6.0 avec ses équivalents dans Visual Basic 2005.

L'utilisation des couleurs dans Visual Basic 2005 est semblable à l'utilisation des couleurs dans Visual Basic 6.0 ; toutefois, vous devez savoir qu'il existe quelques différences conceptuelles. De plus, les constantes utilisées pour spécifier des couleurs dans Visual Basic 6.0 sont remplacées par de nouvelles énumérations de couleurs dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, les couleurs étaient représentées par une valeur de type **Long** ; dans Visual Basic 2005, les couleurs sont de type Color. Dans Visual Basic 6.0, les constantes étaient fournies pour huit couleurs standard ; dans Visual Basic 2005 il y a plus de 100 couleurs nommées.

Conseil

Pour rechercher une valeur équivalente à une couleur Visual Basic 6.0 qui n'est pas une couleur standard, vous pouvez utiliser la classe ColorTranslator et lui passer la valeur **Long** de la couleur Visual Basic 6.0.

Color, constantes

Dans Visual Basic 6.0, les constantes étaient fournies pour les couleurs système qui pouvaient être utilisées pour mapper une couleur aux préférences système de l'utilisateur. Dans Visual Basic 2005, les couleurs système sont de type SystemColors.

Propriétés BackColor et ForeColor

Dans Visual Basic 6.0, les propriétés **BackColor** et **ForeColor** d'un contrôle devaient être explicitement définies au moment du design ou au moment de l'exécution, car les couleurs ne pouvaient pas être héritées. Dans Visual Basic 2005, si une couleur n'est pas explicitement définie au moment du design ou au moment de l'exécution, elle hérite des paramètres de couleur de son parent. Pour plus d'informations, consultez Comportement des couleurs pour les utilisateurs de Visual Basic 6.0.

Propriétés Palette et PaletteMode

Dans Visual Basic 6.0, les propriétés **Palette** et **PaletteMode** d'un formulaire étaient utilisées pour définir les couleurs à utiliser lors de l'affichage des images sur un écran 256 couleurs. Visual Basic 2005 ne prend pas en charge les propriétés **Palette** et **PaletteMode**. Pour plus d'informations, consultez Palettes pour les utilisateurs de Visual Basic 6.0.

MaskColor, propriété

La propriété **MaskColor** de Visual Basic 6.0 n'existe plus pour les contrôles CheckBox, Button et RadioButton. Vous pouvez émuler la propriété **MaskColor** à l'aide des méthodes graphiques. Pour plus d'informations, consultez MaskColor pour les utilisateurs de Visual Basic 6.0.

Modifications du code pour la gestion des couleurs

Les exemples suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour affecter la valeur d'une couleur système à une couleur

Le code suivant affecte à la BackColor d'un formulaire une couleur correspondant à celle que l'utilisateur a sélectionnée pour la barre de titre active.

```
' Visual Basic 6.0
Me.BackColor = vbActiveTitleBar
```

VB

```
' Visual Basic 2005
Me.BackColor = System.Drawing.SystemColors.ActiveCaption
```

Modifications du code pour affecter une valeur Visual Basic 6.0 à une couleur

L'exemple suivant affecte à la propriété BackColor d'un contrôle TextBox une couleur correspondant à la nuance de vert la plus claire (`&H00C0FFC0`) du sélecteur de couleurs Visual Basic 6.0.

```
' Visual Basic 6.0
textBox1.Text = &H00C0FFC0
```

VB

```
' Visual Basic 2005
TextBox1.BackColor = System.Drawing.ColorTranslator.FromOle(&HC0FFC0)
```

Équivalences des constantes

Les tableaux suivants répertorient les constantes Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Équivalents des constantes de couleur

Visual Basic 6.0	Équivalent Visual Basic 2005
vbBlack	Black
vbRed	Red
vbGreen	Lime
vbYellow	Yellow
vbBlue	Blue
vbMagenta	Magenta
vbCyan	Cyan
vbWhite	White

Équivalents des constantes de couleur système

Visual Basic 6.0	Équivalent Visual Basic 2005
vb3DDKShadow	ControlDarkDark
vb3DFace	Control
vb3DHighlight	ControlLightLight
vb3DLight	ControlLight
vb3DShadow	ControlDark
vbActiveBorder	ActiveBorder
vbActiveTitleBar	ActiveCaption
vbActiveTitleBarText	ActiveCaptionText
vbApplicationWorkspace	AppWorkspace
vbButtonFace	**Control**
vbButtonShadow	**ControlDark**
vbButtonText	ControlText
vbDesktop	Desktop

vbGrayText	GrayText
vbHighlight	Highlight
vbHighlightText	HighlightText
vbInactiveBorder	InactiveBorder
vbInactiveCaptionText	InactiveCaptionText
vbInactiveTitleBar	InactiveCaption
vbInactiveTitleBarText	**InactiveCaptionText**
vbInfoBackground	Info
vbInfoText	InfoText
vbMenuBar	Menu
vbMenuText	MenuText
vbScrollBars	ScrollBar
vbTitleBarText	**ActiveCaptionText**
vbWindowBackground	Window
vbWindowFrame	WindowFrame
vbWindowText	WindowText

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005 à l'aide de l'Assistant Mise à niveau, les couleurs sont converties à l'aide de la méthode FromOle.

Après la mise à niveau, recherchez tout code qui définit explicitement la propriété **BackColor** ou ForeColor au moment de l'exécution. S'il est défini pour un parent, définissez explicitement la couleur des contrôles enfants au moment du design, sinon, les contrôles enfants héritent des couleurs du parent.

Tout code qui utilisait les propriétés **Palette**, **PaletteMode** ou **MaskColor** devra être réécrit en Visual Basic 2005.

Voir aussi

Concepts

Comportement des couleurs pour les utilisateurs de Visual Basic 6.0
Palettes pour les utilisateurs de Visual Basic 6.0
MaskColor pour les utilisateurs de Visual Basic 6.0

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Comportement des couleurs pour les utilisateurs de Visual Basic 6.0

Les propriétés **ForeColor** et **BackColor** de contrôles se comportent différemment dans Visual Basic 6.0 et dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, les propriétés **BackColor** et **ForeColor** d'un contrôle devaient être explicitement définies au moment du design ou au moment de l'exécution, car les couleurs ne pouvaient pas être héritées. Dans Visual Basic 2005, si une couleur n'est pas explicitement définie au moment du design ou au moment de l'exécution, elle hérite des paramètres de couleur de son parent.

Modifications du code pour le comportement des couleurs

Ces différences de comportement sont illustrées dans l'exemple suivant qui s'appuie sur un formulaire contenant deux contrôles **CommandButton** , `Command1` et `Command2`.

```
' Visual Basic 6.0

' Command1's BackColor is left at its default (gray).

' Command2's BackColor is explicitly set.

Command2.BackColor = vbBlack

' Explicitly set the BackColor of the form.

Form1.BackColor = vbRed
```

Après exécution du code ci-dessus, **BackColor** de `Command1` reste le gris (valeur par défaut) et **BackColor** de `Command2` reste le noir.

L'exemple suivant illustre le comportement dans Visual Basic 2005. Dans Visual Basic 2005, les contrôles **CommandButton** sont remplacés par les contrôles Button.

VB

```
' Visual Basic 2005
' Command1's BackColor is left at its default (gray).
' Command2's BackColor is explicitly set.
Command2.BackColor = System.Drawing.Color.Black
' Explicitly set the BackColor of the form.
Me.BackColor = System.Drawing.Color.Red
```

Après avoir exécuté le code ci-dessus, BackColor de `Command1` est rouge et **BackColor** de `Command2` est noir. Parce que le **BackColor** de `Command1` n'a pas été défini explicitement, le rouge est hérité du formulaire.

Remarque

Cela s'applique pour tout parent, pas seulement pour les formulaires. Si le contrôle **Button** était contenu dans un contrôle Panel du formulaire, le changement de la couleur du volet ou du formulaire modifierait la couleur de **Button**.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, l'Assistant Mise à niveau n'a aucune façon de déterminer si les couleurs seront héritées. Après la mise à niveau, recherchez tout code qui définit explicitement la propriété **BackColor** ou ForeColor au moment de l'exécution. Si cette propriété est définie pour un parent, définissez explicitement la couleur des contrôles enfants au moment du design.

Voir aussi

Référence

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

MaskColor pour les utilisateurs de Visual Basic 6.0

La propriété **MaskColor** dans Visual Basic 6.0 n'a aucun équivalent direct dans Visual Basic 2005, toutefois vous pouvez utiliser des méthodes graphiques pour obtenir le même effet.

Différences conceptuelles

Dans Visual Basic 6.0, la propriété **MaskColor** d'un contrôle **CheckBox**, **CommandButton** ou **OptionButton** était utilisée pour définir une couleur qui devait devenir transparente, permettant ainsi l'affichage d'une image d'arrière-plan. Pour utiliser cette propriété, la propriété **Style** devait avoir la valeur **Graphical**, la propriété **UseMaskColor** devait avoir la valeur **True** et une bitmap devait être assignée à la propriété **Picture**.

Dans Visual Basic 2005, il n'y a aucun équivalent direct à la propriété **MaskColor**. Cependant, vous pouvez définir la transparence d'un contrôle à l'aide de la méthode MakeTransparent d'un objet Bitmap.

Modifications apportées au code de MaskColor

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

```
' Visual Basic 6.0

' Assumes a picture has been assigned to the Picture property

' and that the Style property has been set to Graphical.

Command1.UseMaskColor = True

Command1.MaskColor = vbWhite
```

VB

```
' Visual Basic 2005
' Assumes a picture has been assigned to the BackgroundImage property.
Dim g As New System.Drawing.Bitmap(Button1.BackgroundImage)
g.MakeTransparent(System.Drawing.Color.White)
Button1.BackgroundImage = g
```

Mettre à niveau des remarques

Lors de la mise à niveau d'une application de Visual Basic 6.0 vers Visual Basic 2005, l'Assistant Mise à niveau ne met pas à niveau le code qui utilise la propriété **MaskColor**. Des avertissements de mise à niveau seront insérés dans votre code. Vous devrez modifier le code avant d'exécuter l'application.

Voir aussi

Référence

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

MakeTransparent

Concepts

Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0

Contrôle CommandButton pour les utilisateurs de Visual Basic 6.0

Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0

Palettes pour les utilisateurs de Visual Basic 6.0

Les propriétés **Palette** et **PaletteMode** dans Visual Basic 6.0 n'ont aucun équivalent dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, les propriétés **Palette** et **PaletteMode** d'un formulaire contrôle les couleurs utilisées lors de l'affichage d'images sur un écran 256 couleurs. Ces propriétés sont sans effet sur les écrans qui prenaient en charge plus de 256 couleurs.

Visual Basic 2005 ne prend pas en charge les propriétés **Palette** ou **PaletteMode**. Les périphériques d'affichage d'aujourd'hui affichent généralement des milliers ou des millions de couleurs et il n'est plus nécessaire de spécifier une palette pour une image.

Remarque
L'objet Image dans Visual Basic 2005 possède une propriété Palette pour restituer des images ; ce n'est pas la même que la p ropriété **Palette** de Visual Basic 6.0 .

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 vers Visual Basic 2005, tout code qui référence la propriété **Palette** ou **PaletteMode** n'est pas mis à niveau. L'Assistant Mise à niveau insère un avertissement dans votre code ; vous devrez supprimer le code qui référence ces propriétés.

Voir aussi

Référence

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0

Bien que les tableaux de contrôles ne soient plus pris en charge dans Visual Basic 2005, vous pouvez dupliquer et étendre un certain nombre de fonctionnalités de tableau de contrôles à l'aide du modèle d'événement.

Différences conceptuelles

Dans Visual Basic 6.0, les tableaux de contrôles pourraient être utilisés pour gérer des contrôles sur un formulaire ; ils fournissaient des fonctions de partage de gestionnaires d'événements en parcourant les groupes de contrôles et en ajoutant les contrôles au moment de l'exécution.

Dans Visual Basic 2005, les tableaux de contrôles ne sont plus pris en charge. Les modifications apportées au modèle d'événement rendent les tableaux de contrôles inutiles, et le .NET Framework fournit les mêmes fonctions pour utiliser les contrôles.

Partage de gestionnaires d'événements

Dans Visual Basic 6.0, les tableaux de contrôles pouvaient être utilisés pour spécifier un groupe de contrôles partageant un ensemble d'événements. Les contrôles devaient être de même type et posséder le même nom.

Visual Basic 2005 permet aux gestionnaires d'événements de gérer des événements à partir de plusieurs contrôles, même les contrôles de noms et de types différents.

Vous pouvez, par exemple, ajouter deux contrôles Button (`Button1` et `Button2`) et un contrôle CheckBox (`CheckBox1`) à un formulaire, puis créer un gestionnaire d'événement pour gérer l'événement Click pour ces trois contrôles :

```
Private Sub MixedControls.Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click, Button2.Click, CheckBox1.Click
```

Parcours des contrôles

La possibilité de parcourir un groupe de contrôles à l'aide de la propriété **Index** était une autre fonctionnalité des tableaux de contrôles Visual Basic 6.0. Par exemple, vous pourriez parcourir le tableau de contrôles en utilisant la propriété **Index** comme une variable de boucle pour effacer le texte de tous les contrôles **TextBox** dans un tableau de contrôles.

Les contrôles Visual Basic 2005 n'ont pas de propriété **Index**, mais vous pouvez toujours parcourir les contrôles sur un formulaire ou un conteneur à l'aide de ControlCollection de la classe Control.

Dans Visual Basic 6.0, les contrôles contenus dans un seul tableau de contrôles pourraient être placés sur plusieurs conteneurs. Par exemple, les contrôles **TextBox** contenus sur deux contrôles **Frame** différents pourraient faire partie du même tableau de contrôles.

Dans Visual Basic 2005, la collection **Controls** retourne seulement des contrôles placés sur un seul conteneur. Vous devez parcourir séparément les contrôles de chaque contrôle conteneur ; pour ce faire, vous pouvez utiliser une fonction récursive.

Ajout de contrôles au moment de l'exécution

Dans Visual Basic 6.0, les contrôles auraient pu être ajoutés à un tableau de contrôles au moment de l'exécution à l'aide de l'instruction **Load**. Les contrôles devaient être du même type que le tableau de contrôles et le tableau de contrôles devait être créé au moment du design avec au moins un élément. Après l'ajout du contrôle, la propriété **Visible** devait avoir la valeur **True**.

Dans Visual Basic 2005, les contrôles sont ajoutés au moment de l'exécution à l'aide du mot clé **New** dans une instruction **Dim**, puis à l'aide de la méthode Add pour le conteneur dans lequel vous souhaitez ajouter le contrôle.

Ajout de gestionnaires d'événements au moment de l'exécution

Dans Visual Basic 6.0, lorsque vous avez ajouté un contrôle à un tableau de contrôles au moment de l'exécution, les nouveaux événements des contrôles ont été automatiquement gérés par les événements du tableau de contrôles.

Dans Visual Basic 2005, vous devez définir des gestionnaires d'événements pour les contrôles ajoutés au moment de l'exécution. Pour ce faire, il suffit d'utiliser l'instruction **AddHandler**.

Modifications du code des tableaux de contrôles

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

Partage de gestionnaires d'événements

L'exemple suivant illustre le partage du gestionnaire d'événements **Change** (TextChanged dans Visual Basic 2005) pour un groupe de trois contrôles TextBox. Dans Visual Basic 2005, la clause **Handles** du gestionnaire d'événements spécifie le contrôle qui sera géré par l'événement. Le gestionnaire d'événements retourne un **Object** générique, donc il doit être casté au type d'objet spécifique (dans ce cas, **TextBox**) que vous souhaitez gérer à l'aide de la méthode **DirectCast**.

```
' Visual Basic 6.0

Private Sub Text1_Change(Index As Integer)

Select Case Index

Case 0

MsgBox("The text in the first TextBox has changed")

Case 1

MsgBox("The text in the second TextBox has changed")

Case 2

MsgBox("The text in the third TextBox has changed")

End Select

End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBoxes_TextChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles TextBox1.TextChanged, _
TextBox2.TextChanged, TextBox3.TextChanged
    Select Case DirectCast(sender, TextBox).Name
        Case TextBox1.Name
            MsgBox("The text in the first TextBox has changed")
        Case TextBox2.Name
            MsgBox("The text in the second TextBox has changed")
        Case TextBox3.Name
            MsgBox("The text in the third TextBox has changed")
    End Select
End Sub
```

Remarque

Le comportement de l'événement est légèrement différent dans Visual Basic 2005. L'événement **TextChanged** est déclench é lorsque chaque contrôle est initialisé (par exemple dans l'événement `Form_Load`), ainsi que lorsque le texte est modifié au moment de l'exécution. Dans Visual Basic 6.0, l'événement **Change** a été déclenché uniquement une fois le texte modifié.

Parcours des contrôles

L'exemple suivant illustre une fonction pour parcourir un groupe de contrôles zone de texte et effacer leur texte. Dans l'exemple Visual Basic 6.0, la propriété **Index** d'un tableau de contrôles est utilisée comme une variable de boucle.

Dans Visual Basic 2005, un objet **Control** est passé comme un argument ; il contient une collection **ControlCollection** qui inclut tous les contrôles placés sur ce contrôle. L'opérateur **Typeof** est utilisé pour déterminer si chaque contrôle est de type **TextBox**.

Remarque

Un objet Form est de type **Control**; vous pouvez également passer **Form** comme un argument.

Comme les contrôles imbriqués ne sont pas inclus dans la collection **ControlCollection**, la méthode HasChildren est utilisée pour déterminer si chaque contrôle contient d'autres contrôles ; dans le cas échéant, la fonction `ClearText` est appelée de manière récursive.

```
' Visual Basic 6.0

Private Sub ClearText()
```

```
For i = 0 To Text1().UBound

Text1(i).Text = ""

Next

End Sub
```

VB

```
' Visual Basic 2005
Private Sub ClearText(ByVal container As Control)
    Dim ctrl As Control
    For Each ctrl In container.Controls
        If TypeOf (ctrl) Is TextBox Then
            ctrl.Text = ""
        End If
        If ctrl.HasChildren Then
             ClearText(ctrl)
        End If
    Next
End Sub
```

Ajout de contrôles au moment de l'exécution

L'exemple suivant illustre l'ajout d'un contrôle zone de texte à un formulaire au moment de l'exécution. Dans Visual Basic 6.0, le contrôle est ajouté à un tableau de contrôles. Dans Visual Basic 2005, le contrôle est ajouté à la collection **ControlCollection**. Dans Visual Basic 6.0, les événements du nouveau **TextBox** ont été gérés automatiquement par le tableau de contrôles. Dans Visual Basic 2005, vous devez raccorder la gestion des événements via l'instruction **AddHandler**.

Ces deux exemples supposent qu'un contrôle zone de texte est ajouté au formulaire au moment du design, et dans l'exemple Visual Basic 6.0 qu'un tableau de contrôles à un seul élément a été créé. L'exemple Visual Basic 2005 suppose également qu'un gestionnaire d'événements nommé `TextChangedHandler` existe pour le premier contrôle **TextBox**.

```
' Visual Basic 6.0

Private Sub AddControl()

' Add a TextBox as the second element of a control array.

Load Text1(1)

' Set the location below the first TextBox.

Text1(1).Move Text1(0).Left, Text1(0).Top + 500

' Make the new TextBox visible

Text1(1).Visible = True
```

VB

```
' Visual Basic 2005
' Declare a new TextBox.
Dim TextBox2 As New TextBox
' Set the location below the first TextBox
TextBox2.Left = TextBox1.Left
TextBox2.Top = TextBox1.Top + 30
' Add the TextBox to the form's Controls collection.
Me.Controls.Add(TextBox2)
AddHandler TextBox2.TextChanged, AddressOf TextChangedHandler
```

Mettre à niveau des remarques

Lorsqu'une application créée avec Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les tableaux de contrôles sont mis à niveau vers les classes spéciales de tableau de contrôles spécifiques au contrôle. Ces classes sont contenues dans l'espace de noms **Microsoft.VisualBasic.Compatibility.VB6** et sont utilisées par les outils de mise à niveau pour émuler le comportement du tableau de contrôles Visual Basic 6.0.

Bien qu'il soit possible d'utiliser ces classes de tableau de contrôles dans un nouveau développement de Visual Basic 2005, nous vous recommandons d'utiliser à la place le modèle d'événement et les fonctions du .NET Framework.

Voir aussi

Concepts

Événements et gestion des événements pour les utilisateurs Visual Basic 6.0

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Modifications des boîtes de dialogue pour les utilisateurs de Visual Basic 6.0

Les méthodes permettant d'afficher des boîtes de dialogue dans Visual Basic 2005 diffèrent de celles de Visual Basic 6.0.

Différences conceptuelles

Dans Visual Basic 6.0, un formulaire peut être affiché en tant que boîte de dialogue modale en appelant la méthode **Show** avec un paramètre **vbModal**.

Dans Visual Basic 2005, la méthode ShowDialog permet l'affichage modal d'un formulaire ; la méthode **Show** permet l'affichage non modal d'un formulaire

En outre, le contrôle **CommonDialog** de Visual Basic 6.0 propose plusieurs boîtes de dialogue prédéfinies. Dans Visual Basic 2005, ce contrôle est remplacé par le contrôle ColorDialog, le contrôle FontDialog, le contrôle OpenFileDialog, le contrôle PageSetupDialog, le contrôle PrintDialog, le contrôle PrintPreviewDialog et le contrôle SaveFileDialog.

Voir aussi

Concepts

Contrôle CommonDialog pour les utilisateurs de Visual Basic 6.0

Autres ressources

Boîtes de dialogue dans les Windows Forms

Contrôles et composants de boîte de dialogue (Windows Forms)

Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0

Le modèle permettant d'implémenter l'édition par glisser-déplacer dans Visual Basic 2005 est totalement différent du modèle Visual Basic 6.0.

Différences conceptuelles

Dans Visual Basic 6.0, l'édition par glisser-déplacer peut s'effectuer de deux méthodes différentes : le glissement standard, qui permet de faire glisser des éléments entre les contrôles sur un formulaire, et le glissement OLE, qui permet de faire glisser des éléments entre formulaires et applications.

Dans Visual Basic 2005, un seul modèle d'édition par glisser-déplacer est utilisé. Il est semblable au glissement OLE, mais les noms et les paramètres des méthodes et des événements de glisser-déplacer sont différents, tout comme le modèle d'événement.

Modifications du code pour la fonction de glisser-déplacer

Modifications du code pour glisser-déplacer du texte

L'exemple suivant montre les différences de code pour déplacer du texte d'un contrôle **TextBox** à un autre :

```
' Visual Basic 6.0
Private Sub Text1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Text1.Text = "Hello World"
    ' Begin dragging by calling the OLEDrag method.
    Text1.OLEDrag
End Sub

Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)
    ' Only allow copying.
    AllowedEffects = vbDropEffectCopy
    Data.Clear
    ' Populate the Data object with the text to copy and the format.
    Data.SetData Text1.Text, vbCFText
End Sub

Private Sub Text2_OLEDragOver(Data As DataObject, Effect As Long, Button As Integer, Shift
As Integer, X As Single, Y As Single, State As Integer)
    ' Make sure that the format is text.
    If Data.GetFormat(vbCFText) Then
      ' If it is text, enable dropping for the second TextBox.
      Text2.OLEDropMode = vbOLEDropManual
    End If
End Sub

Private Sub Text2_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift
As Integer, X As Single, Y As Single)
    ' Copy the text from the Data object to the second TextBox.
    Text2.Text = Data.GetData(vbCFText)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load

    ' Dropping must be enabled before the dragging occurs.
    TextBox2.AllowDrop = True
End Sub

Private Sub TextBox1_MouseDown(ByVal sender As Object, ByVal e _
As System.Windows.Forms.MouseEventArgs) Handles TextBox1.MouseDown
```

```
    TextBox1.Text = "Hello World"
    ' Begin dragging by calling the DoDragDrop method.
    ' OLEStartDrag is replaced by arguments on the method.
    TextBox1.DoDragDrop(TextBox1.Text, DragDropEffects.Copy)
End Sub

Private Sub TextBox2_DragEnter(ByVal sender As Object, ByVal e _
As System.Windows.Forms.DragEventArgs) Handles TextBox2.DragEnter

    ' Make sure that the format is text.
    If (e.Data.GetDataPresent(DataFormats.Text)) Then
      ' Allow drop.
      e.Effect = DragDropEffects.Copy
    Else
      ' Do not allow drop.
      e.Effect = DragDropEffects.None
    End If
End Sub

Private Sub TextBox2_DragDrop(ByVal sender As Object, ByVal e _
As System.Windows.Forms.DragEventArgs) Handles TextBox2.DragDrop

    ' Copy the text to the second TextBox.
    TextBox2.Text = e.Data.GetData(DataFormats.Text).ToString
End Sub
```

Mettre à niveau des remarques

Le code glisser-déplacer ne peut pas être automatiquement mis à niveau vers Visual Basic 2005 ; vous devez le réécrire en utilisant le nouveau modèle. Tout code glisser-déplacer est marqué par des avertissements de mise à niveau pendant le processus de mise à niveau.

Voir aussi

Autres ressources

Opérations glisser-déplacer et prise en charge du Presse-papiers

Échange dynamique de données pour les utilisateurs de Visual Basic 6.0

L'échange dynamique de données (DDE, Dynamic Data Exchange) n'est plus pris en charge dans Visual Basic 2005.

Différences conceptuelles

L'échange dynamique de données (DDE, Dynamic Data Exchange) était une technologie qui permettait aux applications de s'échanger des informations dans Visual Basic 3.0. Cette technologie a été remplacée par OLE Automation dans les versions ultérieures de Visual Basic ; toutefois, pour des raisons de compatibilité descendante, elle a toujours été prise en charge jusqu'à la version Visual Basic 6.0.

Avec l'introduction de Visual Basic .NET 2002, le DDE n'est plus pris en charge. Les applications Visual Basic devant échanger des informations avec les applications héritées (legacy) qui prennent en charge le DDE uniquement doivent être conservées dans Visual Basic 6.0.

Les propriétés, méthodes et événements DDE suivants ne sont plus disponibles :

Catégorie	**Nom**
Propriétés	**LinkItem** **LinkMode** **LinkTimeOut** **LinkTopic**
Méthodes	**LinkExecute** **LinkPoke** **LinkRequest** **LinkSend**
Événements	**LinkClose** **LinkError** **LinkExecute** **LinkNotify** **LinkOpen**

En outre, la constante **vbCFLink** de l'objet **Clipboard** n'est plus valide.

Mettre à niveau des remarques

Lors de la mise à niveau d'une application Visual Basic 6.0 vers Visual Basic 2005, tout code qui fait référence à des propriétés ou méthodes DDE et tout code dans les procédures événementielles DDE ne sera pas mis à niveau et un avertissement sera ajouté.

Voir aussi

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0
Contrôle Label pour les utilisateurs de Visual Basic 6.0
Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0
Contrôle TextBox pour les utilisateurs de Visual Basic 6.0
Objet Clipboard pour les utilisateurs de Visual Basic 6.0

Gestion des polices pour les utilisateurs de Visual Basic 6.0

Cette rubrique compare des techniques de gestion des polices dans Visual Basic 6.0 avec ses équivalents dans Visual Basic 2005.

Différences conceptuelles

Dans Visual Basic 6.0, les polices sont gérées de deux façons : en tant que propriétés de la police des formulaires et des contrôles ou en tant qu'objet **stdFont**.

Dans Visual Basic 2005, il existe un seul objet **Font** : **System.Drawing.Font**. La propriété **Font** d'un formulaire ou d'un contrôle prend un objet **Font** comme un argument.

Définition des propriétés de police

Dans Visual Basic 6.0, les propriétés de police peuvent être définies au moment de l'exécution, soit en assignant un objet **stdFont** ou en définissant directement les propriétés sur le contrôle. Les deux méthodes sont interchangeables.

Dans Visual Basic 2005, la propriété **Font** d'un contrôle est en lecture seule au moment de l'exécution. Vous ne pouvez pas définir les propriétés directement. Vous devez instancier un nouvel objet **Font** chaque fois que vous souhaitez définir une propriété.

Héritage des polices

Dans Visual Basic 6.0, les propriétés de la police doivent être définies individuellement pour chaque contrôle ou formulaire ; l'utilisation d'un objet **stdFont** simplifie le processus, mais nécessite encore du code.

Dans Visual Basic 2005, les propriétés de la police sont automatiquement héritées de leur parent, sauf lorsqu'elles sont explicitement définies pour l'objet enfant. Par exemple, si vous avez deux contrôles label sur un formulaire et modifiez la propriété de police du formulaire en Arial, la police du contrôle label change également en Arial. Si vous modifiez ensuite la police d'une étiquette en Times Roman, toute modification supplémentaire de la police du formulaire ne substituera pas la police de l'étiquette.

Compatibilité des polices

Dans un souci de compatibilité descendante, Visual Basic 6.0 prend en charge des polices Raster ; Visual Basic 2005 prend uniquement en charge les polices TrueType et OpenType.

Énumération des polices

Dans Visual Basic 6.0, vous pouvez utiliser la collection **Screen.Fonts** avec la propriété **Screen.FontCount** pour énumérer les polices écran disponibles.

Dans Visual Basic 2005, l'objet **Screen** n'existe plus ; pour énumérer les polices disponibles dans le système, vous devez utiliser l'espace de noms **System.Drawing.FontFamily**.

Remarque
Visual Basic 6.0 énumère tous les types de polices. Visual Basic 2005 prend uniquement en charge les polices TrueType et OpenType ; les autres types de police ne sont pas énumérés. En outre, Visual Basic 6.0 énumère chaque version de jeu de caractères au sein d'une famille de polices (par exemple, Arial, Arial Baltic, Arial Greek) ; Visual Basic 2005 énumère seulement les familles de polices.

Modifications du code pour les polices

Les exemples de code suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour définir les propriétés Font

L'exemple suivant illustre la définition des propriétés de la police au moment de l'exécution. Dans Visual Basic 6.0, vous pouvez définir directement des propriétés sur un contrôle ; dans Visual Basic 2005, vous devez créer un nouvel objet **Font** et l'assigner au contrôle chaque fois que vous devez définir une propriété.

```
' Visual Basic 6.0
' Set font properties directly on the control.
Label1.FontBold = True
' Create a stdFont object.
```

```
Dim f As New stdFont
' Set the stdFont object to the Arial font.
f.Name = "Arial"
' Assign the stdFont to the control's font property.
Set Label1.Font = f
' You can still change properties at run time.
Label1.FontBold = True
Label1.FontItalic = True
```

VB

```
' Visual Basic 2005
' Create a new Font object  Name and Size are required.
Dim f As New System.Drawing.Font("Arial", 10)
' Assign the font to the control
Label1.Font = f
' To set additional properties, you must create a new Font object.
Label1.Font = New System.Drawing.Font(Label1.Font, FontStyle.Bold Or FontStyle.Italic)
```

Modifications du code pour énumérer des polices

L'exemple suivant illustre le remplissage d'un contrôle **ListBox** avec une liste des polices installées sur un ordinateur.

Remarque

Visual Basic 6.0 énumère tous les types de polices. Visual Basic 2005 prend uniquement en charge les polices TrueType et OpenType ; les autres types de police ne sont pas énumérés. En outre, Visual Basic 6.0 énumère chaque version de jeu de caractères au sein d'une famille de polices (par exemple, Arial, Arial Baltic, Arial Greek) ; Visual Basic 2005 énumère seulement les familles de polices.

```
' Visual Basic 6.0
Dim i As Integer
For i = 0 To Screen.FontCount – 1
   List1.AddItem Screen.Fonts(i)
Next i
```

VB

```
' Visual Basic 2005
Dim ff As FontFamily
For Each ff In System.Drawing.FontFamily.Families
  listBox1.Items.Add(ff.Name)
Next
```

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tout code de gestion des polices est modifié afin d'utiliser le nouvel objet **Font**.

L'héritage des polices dans Visual Basic 2005 peut provoquer des modifications involontaires de l'apparence de votre application. Vous devez rechercher dans votre application convertie tout code définissant explicitement une police au niveau du formulaire ou du conteneur et, si nécessaire, changer la police si des contrôles enfants ne doivent pas en hériter.

Lors de la mise à niveau, les polices Raster sont converties dans la police par défaut OpenType, Microsoft Sans Serif. Les caractéristiques de mise en forme telles que le gras ou l'italique ne sont pas conservées. Pour plus d'informations, consultez Seules les polices OpenType et TrueType sont prises en charge.

Si votre application contient le code qui énumère des polices, les polices Raster ne sont pas énumérées dans l'application mise à niveau et les familles de polices sont énumérées au lieu des versions de jeu de caractères.

Voir aussi

Référence

Font Class

FontFamily.Families Property

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Positionnement de formulaire pour les utilisateurs de Visual Basic 6.0

La définition de la position d'un formulaire à l'écran est différente entre Visual Basic 6.0 et Visual Basic 2005.

Différences conceptuelles

Propriétés Left et Top

Dans Visual Basic 6.0, l'emplacement initial d'un formulaire à l'écran peut être défini au moment du design à l'aide de la fenêtre **Présentation du formulaire** ou en définissant les propriétés **Left** et **Top** dans la fenêtre **Propriétés**. Les propriétés **Left** et **Top** peuvent également être définies par programme.

Dans Visual Basic 2005, l'emplacement initial à l'écran est défini au moment du design par la définition de la propriété **Location**, ou par programme par l'affectation d'un objet **Point** à la propriété **Location**.

Propriété StartUpPosition

De plus, Visual Basic 6.0 prend en charge une propriété **StartUpPosition** qui peut être utilisée pour permettre à Windows de définir automatiquement un emplacement (par exemple, centré à l'écran). Le paramètre par défaut est **Manual**, ce qui signifie que les propriétés **Left** et **Top** ont priorité.

Dans Visual Basic 2005, la propriété **StartPosition** exécute la même fonction. Le paramètre par défaut est à présent **WindowsDefaultLocation**, c'est-à-dire que la propriété **Location** est ignorée.

Voir aussi

Tâches

Comment : définir l'emplacement à l'écran des Windows Forms

Tâches des formulaires pour les utilisateurs de Visual Basic 6.0

Il existe de nombreuses différences entre les Windows Forms et les formulaires Visual Basic 6.0. Le tableau suivant répertorie les tâches qui peuvent être accomplies avec les formulaires et présente leur méthodologie. Vous pouvez vous servir de cette liste pour opérer la transition entre vos projets de formulaires Visual Basic 6.0 et le modèle .NET Framework des Windows Forms.

Tâche	Formulaires Visual Basic 6.0	Windows Forms
Définissez l'indexation d'onglet pour tous les contrôles sur un formulaire.	Sélectionnez chaque contrôle du formulaire et définissez sa propriété **TabIndex** en indiquant comme valeur le numéro suivant dans l'ordre de tabulation du formulaire.	Dans le menu **Affichage**, choisissez **Ordre de tabulation**. Le formulaire affiche en regard de chaque contrôle un chiffre représentant la valeur de la propriété TabIndex du contrôle. Cliquez sur chaque contrôle dans l'ordre de tabulation voulu. Pour plus d'informations, consultez Comment : définir l'ordre de tabulation dans les Windows Forms.
Détermination de la touche de modification (ALT, MAJ, CTRL) devant être enfoncée durant les événements de clavier et de souris.	Utilisez les paramètres fournis par le gestionnaire d'événements de la souris.	Comparez la méthode ModifierKeys de la classe du formulaire avec la valeur de la touche de modification dans l'énumération Keys. Pour plus d'informations, consultez Comment : identifier la touche de modification activée.
Alignement ou dimensionnement des contrôles en fonction de l'emplacement et de la taille d'un seul contrôle.	Cliquez sur chaque contrôle à aligner ou à dimensionner. Le dernier contrôle sélectionné sert à déterminer l'emplacement et la taille de tous les contrôles sélectionnés.	Cliquez sur chaque contrôle à aligner ou à dimensionner. Le premier contrôle sélectionné sert à déterminer l'emplacement et la taille de tous les contrôles sélectionnés. Pour plus d'informations, consultez Comment : aligner plusieurs contrôles dans des Windows Forms.
Modification du nom d'un formulaire.	Modifiez le nom spécifié dans la propriété **Name** du formulaire, dans la fenêtre **Propriétés**.	Cliquez sur le fichier de classe du formulaire dans l'**Explorateur de solutions**, puis modifiez la propriété **Filename** dans la fenêtre **Propriétés**.
Modification de la position d'affichage du formulaire au moment de l'exécution.	Utilisez la fenêtre **Présentation des formulaires** pour positionner le formulaire ou bien définissez sa propriété **Startup** dans la fenêtre **Propriétés**.	Une fois le formulaire sélectionné dans le **Concepteur Windows Forms**, définissez la propriété StartPosition dans la fenêtre **Propriétés**.
Affichage d'un formulaire en tant que boîte de dialogue	Dans le code, utilisez la méthode **Show** et spécifiez la valeur **vbModal** pour indiquer que l'affichage du formulaire doit être modal.	Dans le code, utilisez la méthode ShowDialog du formulaire. Pour plus d'informations, consultez Comment : afficher des boîtes de dialogue pour les Windows Forms.
Groupement ou superposition de contrôles dans un formulaire.	Superposez les contrôles dans le contrôle **Frame**. Pour déplacer les contrôles sous la forme d'un groupe; sélectionnez-les tous.	Placez les contrôles à l'intérieur d'un contrôle GroupBox. Le déplacement de la zone de groupe a pour effet de déplacer également les contrôles.
Itération dans tous les contrôles d'un formulaire.	Utilisez un groupe de contrôles.	Utilisez Controls Collection. Pour plus d'informations, consultez Collections en Visual Basic.

Voir aussi

Référence

Vue d'ensemble des Windows Forms

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0
Tableaux de contrôles destinés aux utilisateurs de Visual Basic 6.0

Autres ressources

Concepts Windows Forms pour les utilisateurs de Visual Basic 6.0

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, plusieurs méthodes et propriétés graphiques sont utilisées pour dessiner sur un contrôle **Form** ou **PictureBox**. Dans Visual Basic 6.0, les graphismes sont basés sur les API interface GDI (Graphics Device Interface) de Windows.

Dans Visual Basic 2005, les graphiques sont fournis par l'espace de noms System.Drawing qui encapsule des API GDI (Graphics Device Interface +). GDI+ développe les possibilités graphiques de Visual Basic 6.0, mais les méthodes ne sont pas compatibles.

Différences conceptuelles

Dans Visual Basic 6.0, les méthodes graphiques s'appliquent uniquement à l'objet **Form** et au contrôle **PictureBox**.

Dans Visual Basic 2005, les méthodes graphiques s'appliquent à des formulaires et à tout contrôle qui prend en charge l'événement Paint, y compris les contrôles PictureBox, Panel et GroupBox. En outre, les méthodes graphiques s'appliquent à tous les contrôles qui prennent en charge la propriété OwnerDraw, y compris les contrôles ListView, TreeView et Button.

Propriété AutoRedraw

Dans Visual Basic 6.0, les méthodes graphiques peuvent être appelées à partir de toutes les procédures événementielles ; la propriété **AutoRedraw** est utilisée pour rendre des graphiques persistants lorsque les méthodes graphiques sont appelées d'un événement autre que **Paint**.

Dans Visual Basic 2005, les méthodes graphiques doivent être appelées uniquement depuis la procédure événementielle **Paint**, ou pour certains contrôles owner-drawn, depuis plusieurs procédures événementielles **Draw** (DrawItem, DrawSubItem, etc.). La propriété **AutoRedraw** n'est plus prise en charge et n'est pas nécessaire parce que les événements **Paint** et **Draw** rendre automatiquement les graphiques persistants.

Propriété ClipControls

Dans Visual Basic 6.0, la propriété **ClipControls** est utilisée pour contrôler la peinture d'un formulaire ou d'un contrôle. Si **True**, seules les zones récemment exposées sont repeintes, en théorie, avec une amélioration des performances.

Il n'existe aucun équivalent pour la propriété **ClipControls** dans Visual Basic 2005 ; l'amélioration des performances dans GDI+ et les cartes vidéos à jour la rendent inutile.

Propriété DrawMode

Dans Visual Basic 6.0, la propriété **DrawMode** contrôle la couleur d'un objet Graphics lorsqu'il dessine un modèle sur un autre. Cette propriété affecte des affichages monochromes ou de faible résolution (256 couleurs ou moins).

Il n'existe aucun équivalent pour la propriété **DrawMode** dans Visual Basic 2005 ; ils ne sont plus nécessaires avec les affichages actuels.

Propriété DrawStyle

Dans Visual Basic 6.0, la propriété **DrawStyle** contrôle l'apparence d'une ligne dessinée à l'aide de la méthode **Line**. Si la propriété **DrawWidth** a une valeur supérieure à 1, la propriété **DrawStyle** n'a aucun effet et la ligne sera toujours unie.

Dans Visual Basic 2005, l'apparence d'une ligne est contrôlée par la définition de la propriété DashStyle d'une classe **System.Drawing.Pen** utilisé par l'une des méthodes **DrawLine** ; la largeur de ligne n'a aucune portée sur cette propriété.

Propriété DrawWidth

Dans Visual Basic 6.0, la propriété **DrawWidth** détermine l'épaisseur d'une ligne en pixels ; la propriété **DrawWidth** est généralement définie avant l'exécution d'une méthode graphique.

Dans Visual Basic 2005, la propriété **Pen.Width** d'un contrôle **System.Drawing.Pen** détermine l'épaisseur de ligne ; vous pouvez définir la propriété **Width** comme un paramètre lorsque vous créez **Pen**, ou en définissant **Pen.Width** une fois **Pen** créé. Si aucune propriété **Pen.Width** n'est spécifiée, la largeur par défaut est de 1 pixel.

Propriété Image

Dans Visual Basic 6.0 la propriété **Image** d'un formulaire ou d'un contrôle **PictureBox** retourne un handle à une bitmap ; le handle peut être assigné à la propriété **Picture** ou utilisé comme valeur pour passer aux appels API Windows.

Dans Visual Basic 2005, les bitmaps ne possèdent plus de handle ; la bitmap réelle est passée comme un objet de type Bitmap. Un contrôle **Bitmap** peut être assigné à la propriété Image d'un contrôle **PictureBox**, mais il ne peut pas être passé aux appels API Windows.

Méthode Line

Dans Visual Basic 6.0, la méthode **Line** est utilisée pour dessiner un rectangle en spécifiant les coordonnées des coins supérieur et inférieur gauches, avec un argument `B` optionnel. La propriété **FillColor** est utilisée pour remplir un rectangle d'une couleur unie, et la propriété **FillStyle** remplit le rectangle de hachures croisées.

Dans Visual Basic 2005, la méthode DrawRectangles est utilisée pour dessiner la bordure d'un rectangle, et la méthode FillRectangle est utilisée pour le remplir. **FillRectangle** utilise un objet Brush comme un paramètre. SolidBrush remplace la propriété **FillColor** et les membres de la classe HatchBrush remplacent la propriété **FillStyle**.

Méthode Point

Dans Visual Basic 6.0, la méthode **Point** d'un formulaire ou d'un contrôle **PictureBox** est utilisée pour retourner une valeur de couleur pour le pixel à un point spécifié. Bien que la méthode **Point** puisse être utilisée pour les formulaires ou les contrôles qui ne contiennent pas d'image, elle est utilisé plus souvent pour récupérer une couleur d'une bitmap assignée à la propriété **Picture**.

Dans Visual Basic 2005, la méthode **Point** n'existe plus. Vous pouvez utiliser la méthode **M:System.Drawing.Bitmap.GetPixel(System.Int32,System.Int32)** pour récupérer une valeur de couleur d'une bitmap. Pour les formulaires ou les contrôles qui ne contiennent pas d'image, vous pouvez interroger la propriété BackColor.

Méthode Print

Dans Visual Basic 6.0, la méthode **Print** est utilisée pour afficher du texte sur un formulaire ou un contrôle **PictureBox**. La police utilisée pour afficher le texte est déterminée par les propriétés **Font** du formulaire ou du contrôle, et la couleur est déterminée par la propriété **ForeColor**. La méthode **Print** n'offre aucun contrôle de l'emplacement du texte et ne peut afficher le texte qu'horizontalement.

Dans Visual Basic 2005, la méthode **DrawString** est utilisée pour afficher du texte. La police est déterminée par un objet Font et la couleur par un objet Brush ; tous les deux sont passés comme paramètres à la méthode **DrawString**. La méthode **DrawString** dispose également des paramètres *X* et *Y* qui déterminent l'emplacement de début du texte. Il existe également un paramètre *Format* optionnel qui utilise un objet StringFormat vous permettant d'afficher le texte verticalement.

Méthode PSet

Dans Visual Basic 6.0, la méthode **PSet** est utilisée pour modifier la couleur d'un pixel sur un formulaire ou un contrôle **PictureBox**. Si la propriété **DrawWidth** a une valeur supérieure à 1, la méthode **PSet** dessine un cercle plein. Un paramètre optionnel est utilisé pour spécifier la couleur ; s'il est omis, le paramètre **ForeColor** est utilisé.

Dans Visual Basic 2005, il n'existe aucun équivalent de la méthode **PSet**. Pour modifier la couleur d'un seul pixel sur un formulaire ou sur le contrôle **PictureBox**, utilisez la méthode **DrawEllipse** pour dessiner un cercle avec une hauteur et une largeur de 1 pixel. Pour dupliquer les fonctionnalités de **PSet** lorsque **DrawWidth** est supérieur à 1, utilisez la méthode **FillEllipse**.

Modifications des codes graphiques

Les exemples de code suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Dessin d'une courbe

Le code suivant montre comment dessiner une ligne sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, la méthode **Line** est utilisée; elle utilise comme arguments les coordonnées X et Y des points de début et de fin et, éventuellement, une couleur. L'exemple Visual Basic 2005 utilise la méthode **DrawLine** qui utilise comme arguments un objet Pens et les coordonnées X et Y des points de début et de fin.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspond à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a solid black line 200 twips from the top of the form.
    Line (0, 200) - (ScaleWidth, 200), vbBlack
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint(ByVal sender As Object, ByVal e _
```

```
As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a solid black line 25 pixels from the top of the form.
    e.Graphics.DrawLine(Pens.Black, 0, 25, Me.Width, 25)
End Sub
```

Dessin d'une ligne en pointillés

Le code suivant montre comment dessiner une ligne en pointillés sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, la propriété **DrawStyle** détermine l'apparence de la ligne. L'exemple Visual Basic 2005 utilise un objet Pen, définissant la propriété **DashStyle** pour déterminer l'apparence.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspond à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a dotted line 200 twips from the top of the form.
    Me.DrawStyle = vbDot
    Line (0, 200) - (ScaleWidth, 200), vbBlack
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint1(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a dotted black line 25 pixels from the top of the form.
    Dim LPen As New System.Drawing.Pen(System.Drawing.Color.Black)
    LPen.DashStyle = Drawing2D.DashStyle.Dot
    e.Graphics.DrawLine(LPen, 0, 25, Me.Width, 25)
End Sub
```

Contrôle de l'épaisseur de ligne

Le code suivant montre comment dessiner des lignes de différentes épaisseurs sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, la propriété **DrawWidth** est utilisée. L'exemple Visual Basic 2005 utilise la propriété **Width** de l'objet **Pens**.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a line with a thickness of 1 pixel.
    DrawWidth = 1
    Line (0, 200)-(ScaleWidth, 200), vbBlack
    ' Draw a line with a thickness of 5 pixels.
    DrawWidth = 5
    Line (0, 400)-(ScaleWidth, 400), vbBlack
    ' Draw a line with a thickness of 10 pixels.
    DrawWidth = 10
    Line (0, 600)-(ScaleWidth, 600), vbBlack
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint2(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a line with a thickness of 1 pixel.
    Dim TPen As New System.Drawing.Pen(System.Drawing.Color.Black, 1)
    e.Graphics.DrawLine(TPen, 0, 25, Me.Width, 25)
```

```
    ' Draw a line with a thickness of 5 pixels.
    TPen.Width = 5
    e.Graphics.DrawLine(TPen, 0, 50, Me.Width, 50)
    ' Draw a line with a thickness of 10 pixels.
    TPen.Width = 10
    e.Graphics.DrawLine(TPen, 0, 75, Me.Width, 75)
End Sub
```

Dessin d'un cercle

Le code suivant montre comment dessiner un cercle sur un formulaire au moment de l'exécution. Dans l'exemple Visual Basic 6.0, la méthode **Circle** est utilisée ; elle utilise comme arguments les coordonnées X et Y du centre, du rayon et, éventuellement, une couleur. L'exemple Visual Basic 2005 utilise la méthode **DrawEllipse** qui utilise un objet **Pen**, les coordonnées X et Y du coin supérieur gauche du rectangle englobant ainsi que la largeur et la hauteur comme arguments.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspond à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a 1000 twip diameter red circle
    Circle (500, 500), 500, vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint3(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a 70 pixel diameter red circle.
    e.Graphics.DrawEllipse(Pens.Red, 0, 0, 70, 70)
End Sub
```

Dessin d'un rectangle plein

Le code suivant montre comment dessiner deux rectangles sur un formulaire au moment de l'exécution, un avec un remplissage uni et l'autre avec un des hachures croisées. Dans l'exemple Visual Basic 6.0, les propriétés **FillColor** et **FillStyle** sont utilisées en même temps que la méthode **Line**. Un appel à la méthode **Line** à l'aide du paramètre *B* permet de dessiner un rectangle.

L'exemple Visual Basic 2005 utilise la méthode **Graphics.Rectangle** pour dessiner le cadre, et la méthode **Graphics.FillRectangle** qui utilise un objet **Brush** comme un argument. Dans cet exemple, les contrôles **SolidBrush** et **HatchBrush** sont utilisés.

Remarque

Dans Visual Basic 6.0, l'unité de mesure par défaut correspond à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a solid red rectangle.
    FillColor = vbRed
    FillStyle = vbSolid
    Line (10, 10)- (1000, 500), vbRed, B
    ' Draw a rectangle filled with a crosshatch pattern.
    FillColor = vbBlack
    FillStyle = vbCross
    Line (10, 500)- (1000, 1000), vbBlack, B
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint4(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a solid red rectangle.
    Dim SBrush As New System.Drawing.SolidBrush _
      (System.Drawing.Color.Red)
    e.Graphics.DrawRectangle(Pens.Red, 2, 2, 70, 40)
    e.Graphics.FillRectangle(SBrush, 2, 2, 70, 40)

    ' Draw a rectangle filled with a crosshatch pattern.
    Dim HBrush As New System.Drawing.Drawing2D.HatchBrush( _
      System.Drawing.Drawing2D.HatchStyle.Cross, _
      System.Drawing.Color.Black, System.Drawing.Color.Transparent)
    e.Graphics.DrawRectangle(Pens.Black, 2, 40, 70, 40)
    e.Graphics.FillRectangle(HBrush, 2, 40, 70, 40)
End Sub
```

Affichage d'une image sur un formulaire

Le code suivant illustre des méthodes graphiques permettant d'afficher une image sur un formulaire au moment de l'exécution. L'exemple Visual Basic 6.0 utilise la méthode **PaintPicture**. L'exemple Visual Basic 2005 utilise la méthode **DrawImage**.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Create a stdPicture object.
    Dim Pict1 As New stdPicture
    Pict1 = LoadPicture("C:\Windows\Greenstone.bmp")
    PaintPicture Pict1, 0, 0
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint5(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Create a Bitmap object.
    Dim Pict1 As New Bitmap("C:\Windows\Greenstone.bmp")
    e.Graphics.DrawImage(Pict1, 0, 0)
End Sub
```

Affichage de texte dans un formulaire

Le code suivant illustre des méthodes graphiques permettant d'afficher une chaîne de texte sur un formulaire au moment de l'exécution. L'exemple Visual Basic 6.0 utilise la méthode **Print**. L'exemple Visual Basic 2005 utilise la méthode **DrawString**.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Me.Font.Size = 24
    Me.Font.Bold = True
    Me.ForeColor = vbRed
    Print "Hello World!"
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint6(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
```

```
    Dim TextFont As New System.Drawing.Font("Arial", 24, FontStyle.Bold)
    Dim TextBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
    e.Graphics.DrawString("Hello World!", TextFont, TextBrush, 10, 10)
    TextFont.Dispose()
    TextBrush.Dispose()
End Sub
```

Définition de la hauteur et de la largeur d'une chaîne

Le code suivant illustre des méthodes graphiques permettant de déterminer la taille d'une chaîne sur un formulaire au moment de l'exécution, et ensuite de dessiner un rectangle pour l'englober. L'exemple Visual Basic 6.0 utilise les méthodes **TextHeight** et **TextWidth**. L'exemple Visual Basic 2005 utilise la méthode MeasureString qui retourne une structure SizeF.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Me.Font.Size = 24
    Me.Font.Bold = True
    Me.ForeColor = vbRed
    Print "Hello World!"
    Line (0, 0)-(TextWidth("Hello World!"), _
TextHeight("Hello World!")), vbBlack, B
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint7(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    Dim TextFont As New System.Drawing.Font("Arial", 24, FontStyle.Bold)
    Dim TextBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
    e.Graphics.DrawString("Hello World!", TextFont, TextBrush, 10, 10)
    Dim TextSize As New System.Drawing.SizeF
    TextSize = e.Graphics.MeasureString("Hello World!", TextFont)
    e.Graphics.DrawRectangle(Pens.Black, 10, 10, TextSize.Width, TextSize.Height)
    TextFont.Dispose()
    TextBrush.Dispose()
End Sub
```

Dessin d'un pixel unique

L'exemple suivant illustre des méthodes graphiques permettant de modifier la couleur d'un seul pixel sur un formulaire au moment de l'exécution. L'exemple Visual Basic 6.0 utilise la méthode **PSet**. L'exemple Visual Basic 2005 utilise la méthode **DrawEllipse** avec les paramètres **Height** et **Width** dont la valeur est `1`.

Remarque Dans Visual Basic 6.0, l'unité de mesure par défaut correspond à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Me.DrawWidth = 1
    PSet (1000, 1000), vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint8(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    e.Graphics.DrawEllipse(Pens.Red, 70, 70, 1, 1)
End Sub
```

Définition de la couleur d'un pixel unique

Le code suivant illustre des méthodes graphiques permettant de déterminer la couleur d'un pixel situé à un emplacement particulier dans une image sur un formulaire au moment de l'exécution, et de dessiner ensuite un rectangle plein de cette couleur. L'exemple Visual Basic 6.0 utilise la méthode **Point** pour récupérer la valeur de couleur. L'exemple Visual Basic 2005 utilise la méthode GetPixel.

Remarque
Dans Visual Basic 6.0, l'unité de mesure par défaut correspond à des twips ; dans Visual Basic 2005, il s'agit de pixels.

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Dim PixelColor As Long
    Picture1.Picture = LoadPicture("C:\Windows\Greenstone.bmp")
    PixelColor = Picture1.Point(10, 10)
    FillColor = PixelColor
    Line (0, 0)-(100, 500), PixelColor, B
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint9(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    Dim Pict1 As New Bitmap("C:\Windows\Greenstone.bmp")
    Picture1.Image = Pict1
    Dim PixelColor As Color = Pict1.GetPixel(4, 4)
    Dim PixelBrush As New SolidBrush(PixelColor)
    e.Graphics.FillRectangle(PixelBrush, 0, 0, 100, 100)
End Sub
```

Équivalences de propriété et de méthode graphiques

Le tableau suivant répertorie les propriétés graphiques et les méthodes de Visual Basic 6.0, ainsi que leurs équivalents Visual Basic 2005.

Visual Basic 6.0	**Équivalent Visual Basic 2005.**
Propriété **AutoRedraw**	Nouvelle implémentation. Pour assurer la persistance des graphismes, placez les méthodes graphiques dans l'événement **Paint**.
Méthode **Circle**	Méthode **DrawEllipse**
Propriété **ClipControls**	Nouvelle implémentation. La propriété **ClipControls** n'est plus nécessaire.
Méthode **Cls**	Méthode **Clear**
Propriété **CurrentX**	Paramètre *x* de plusieurs méthodes graphiques. Par exemple, DrawRectangle (stylet, *x*, y, largeur, hauteur)
Propriété **CurrentY**	Paramètre *y* de plusieurs méthodes graphiques. Par exemple, **DrawRectangle** (stylet, *y*, y, largeur, hauteur)
DrawMode Propriété	Nouvelle implémentation. La propriété **DrawMode** n'est plus nécessaire.
Propriété **DrawStyle**	Propriété **DashStyle**
Propriété **DrawWidth**	Propriété Width

Propriété **FillColor**	Objet **SolidBrush**
Propriété **FillStyle**	Objet **HatchBrush**
Propriété **HasDC**	Nouvelle implémentation. Les contextes de périphérique (DC) ne sont plus nécessaires avec GDI+.
Propriété **HDC**	Nouvelle implémentation. Les contextes de périphérique (DC) ne sont plus nécessaires avec GDI+.
Propriété **Image**	Nouvelle implémentation.
Méthode **Line**	Méthode **DrawLine**
Méthode **PaintPicture**	Méthode **DrawImage**
Méthode **Point**	Pas d'équivalent direct. Pour les bitmaps, utilisez **Bitmap.GetPixel**. Pour les formulaires ou les cont rôles, utilisez la propriété BackColor.
Méthode **Print**	Méthode **DrawString**
Méthode **Pset**	Méthodes **DrawEllipse**, **FillEllipse**
Propriétés **TextHeight**, **T extWidth**	Méthode **MeasureString**

Mettre à niveau des remarques

Lorsqu'une application est mise à niveau de Visual Basic 6.0 à Visual Basic 2005, les méthodes graphiques ne sont pas mises à niveau et des avertissements sont insérés dans le code. En raison des différences considérables entre GDI et GDI+, tout code graphique existant devra être réécrit.

Voir aussi

Tâches

Contrôle utilisateur de dessin personnalisé, exemple

Autres ressources

Vue d'ensemble des graphismes (Windows Forms)
Mise en route de la programmation graphique
À propos du code managé GDI+
Utilisation de classes graphiques managées

Prise en charge de l'aide pour les utilisateurs de Visual Basic 6.0

Cette rubrique compare la prise en charge de l'implémentation de l'aide dans Visual Basic 6.0 avec son équivalent dans Visual Basic 2005.

La plupart des applications fournissent une aide pour les utilisateurs dans le formulaire d'un fichier d'aide, dans l'aide contextuelle et / ou dans les info-bulles. Même si Visual Basic 6.0 et Visual Basic 2005 prennent tous deux en charge les trois mécanismes d'aide, les méthodes d'implémentation de ces mécanismes diffèrent considérablement.

Différences conceptuelles

Dans Visual Basic 6.0, l'aide pouvait être fournie via l'aide HTML ou l'ancien format d'aide Windows. Dans Visual Basic 2005, seule l'aide HTML est prise en charge.

Dans Visual Basic 6.0, la prise en charge de l'aide était implémentée par projet. Pour cela, vous deviez spécifier un nom de fichier d'aide dans la boîte de dialogue **Propriétés du projet**. Chaque formulaire et chaque contrôle possédaient une propriété **HelpContextID** qui pouvait être utilisée pour créer un lien vers une rubrique spécifique du fichier d'aide.

La prise en charge de l'aide dans Visual Basic 2005 est implémentée par formulaire, en ajoutant un ou plusieurs composants HelpProvider à un formulaire. Chaque formulaire et chaque contrôle possèdent des propriétés **HelpKeyword** et **HelpNavigator** utilisées pour créer un lien vers une rubrique spécifique. Pour plus d'informations, consultez Comment : fournir de l'aide dans une application Windows.

Aide contextuelle

Dans Visual Basic 6.0, l'aide contextuelle était implémentée à l'aide des propriétés **WhatsThisButton** et **WhatsThisHelp** d'un formulaire. Le bouton **Qu'est-ce que c'est** apparaissait si **WhatsThisButton** était **True** et si les propriétés **MaxButton** et **MinButton** étaient toutes deux **False**. La définition de la propriété **WhatsThisMode** dans le code activait le bouton.

Dans Visual Basic 2005, l'aide contextuelle est implémentée à l'aide de la propriété HelpButton d'un formulaire. Le bouton **Aide** apparaît uniquement si la propriété **HelpButton** a la valeur **True** et si les propriétés MaximizeBox et MinimizeBox ont toutes deux la valeur **False** ; le bouton est activé automatiquement. Pour plus d'informations, consultez Comment : afficher l'aide contextuelle.

Info-bulles

Dans Visual Basic 6.0, les info-bulles étaient implémentées à l'aide de la propriété **ToolTipText** d'un contrôle.

Dans Visual Basic 2005, les info-bulles sont implémentées en ajoutant un composant ToolTip à un formulaire. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.

Méthode ShowHelp

Dans Visual Basic 6.0, l'aide pourrait également être affichée à l'aide d'une méthode **ShowHelp** d'un contrôle **CommonDialog** pour ouvrir l'aide Windows. L'aide Windows n'est plus prise en charge dans Visual Basic 2005 et il n'existe aucun contrôle équivalent pour afficher l'aide.

Modifications apportées au code de prise en charge de l'aide

Le code suivant illustre les différences dans les techniques de codage entre Visual Basic 6.0 et Visual Basic 2005 pour quelques utilisations communes de propriétés d'aide.

Spécification d'un fichier d'aide

L'exemple suivant illustre comment spécifier un fichier d'aide pour votre application, en supposant que votre formulaire a deux contrôles **OptionButton** qui permettent à l'utilisateur de choisir entre les fichiers d'aide anglais et français.

```
' Visual Basic 6.0

If Option1(0).Value = True Then

App.HelpFile = App.Path & "\EnglishHelp.chm"

Else

App.HelpFile = App.Path & "\FrenchHelp.chm"

End If
```

VB

```
  ' Visual Basic 2005
```

```
' Assumes a HelpProvider component has been added to the form.
If RadioButton1.Checked = True Then
    HelpProvider1.HelpNamespace = My.Application.Info.DirectoryPath & _
        "\EnglishHelp.chm"
Else
    HelpProvider1.HelpNamespace = My.Application.Info.DirectoryPath & _
        "\FrenchHelp.chm"
End If
```

Affichage d'une info-bulle

Les éléments suivants décrivent le code d'affichage d'une info-bulle.

```
' Visual Basic 6.0

Private Sub Text1_Change()

Text1.ToolTipText = "The text has changed"

End Sub
```

VB

```
' Visual Basic 2005
' Assumes a ToolTip component has been added to the form.
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles TextBox1.TextChanged
    ToolTip1.SetToolTip(TextBox1, "The text has changed")
End Sub
```

Équivalences de prise en charge de l'aide

Visual Basic 6.0	Visual Basic 2005
Propriété **App.HelpFile**	Composant **HelpProvider**
Propriété **HelpContextID**	Méthode SetHelpKeyword. Pour plus d'informations, consultez Comment : fournir de l'aide dans une application Windows.
Propriété **ToolTipText**	Méthode SetToolTip. Pour plus d'informations, consultez Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0.
Propriété **WhatsThisButton**	Propriété **HelpButton**
Propriété **WhatsThisHelp**	Propriété **HelpButton**
Propriété **WhatsThisHelpID**	Propriété HelpString
Propriété **WhatsThisMode**	Pas d'équivalent. Activé par défaut.

Mettre à niveau des remarques

Lorsqu'un projet Visual Basic 6.0 est mis à niveau à l'aide de l'Assistant Mise à niveau dans Visual Basic 2005, aucune propriété liée à l'aide ou au code ne sera mise à niveau. Vous devez réimplémenter la prise en charge de l'aide de votre application après la mise à niveau. Si le fichier d'aide a été écrit via l'aide HTML, vous pouvez réutiliser le fichier d'aide ; s'il a été écrit via l'aide Windows, il devra être réécrit.

Voir aussi

Tâches

Comment : fournir de l'aide dans une application Windows
Comment : afficher l'aide contextuelle

Concepts

Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0

Autres ressources

Intégration de l'aide d'utilisateur dans les Windows Forms

Gestion des menus pour les utilisateurs de Visual Basic 6.0

Les techniques de création et d'utilisation des menus dans Visual Basic 2005 diffèrent considérablement de celles de Visual Basic 6.0

Différences conceptuelles

Dans Visual Basic 6.0, les menus étaient créés à l'aide de l'**Éditeur de menus**. Les menus ne pouvaient pas être créés par programme, mais pouvaient toutefois être modifiés ou ajoutés par programme.

Dans Visual Basic 2005, les menus sont créés à l'aide d'un contrôle MenuStrip qui fournit un Concepteur de menus graphiques. Les menus peuvent également être créés par programme.

Menus contextuels

Dans Visual Basic 6.0, les menus contextuels étaient créés par appel à la méthode **PopupMenu** d'un formulaire ou d'un contrôle. Il n'était pas possible de créer les menus par programme ; la méthode **PopupMenu** impliquait l'existence préalable d'un menu.

Dans Visual Basic 2005, les menus contextuels sont créés à l'aide des contrôles ContextMenuStrip qui fournissent un Concepteur de menus graphiques ; ils peuvent également être créés par programme par la création d'une nouvelle instance de la classe **ContextMenuStrip**. Pour plus d'informations, consultez Vue d'ensemble du contrôle ContextMenuStrip.

Fusion de menus

Dans Visual Basic 6.0, la propriété **NegotiateMenus** d'un formulaire déterminait si le menu d'un objet sur le formulaire était fusionné avec le menu du formulaire. Cette propriété ne pouvait être définie qu'au moment du design.

Dans Visual Basic 2005, les menus peuvent être fusionnés à l'aide de la propriété `AllowMerge` d'un contrôle **MenuStrip** ou **ContextMenuStrip**.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, tous les menus existants sont mis à niveau vers des composants MainMenu. Les menus utilisés comme des menus contextuels doivent être modifiés manuellement pour pouvoir utiliser des contrôles **ContextMenuStrip**.

Voir aussi

Concepts

Objet Menu pour les utilisateurs de Visual Basic 6.0

Windows Forms pour les utilisateurs de Visual Basic 6.0

Autres ressources

MenuStrip, contrôle (Windows Forms)

MDI pour les utilisateurs de Visual Basic 6.0

Vous pouvez utiliser à la fois Visual Basic 6.0 et Visual Basic 2005 pour créer des applications d'interface multidocument (MDI); toutefois, les techniques de création et certains de leurs comportements diffèrent.

Différences conceptuelles

Dans Visual Basic 6.0, les applications d'interface multidocument (MDI) sont créées par l'ajout d'un formulaire MDI à un projet puis la définition de la propriété **MDIChild** de tout formulaire enfant.

Visual Basic 2005 n'a pas de formulaire MDI ; tout formulaire peut être transformé en un parent MDI en affectant **True** à la propriété IsMdiContainer.

De nombreuses propriétés et méthodes qui s'appliquaient aux formulaires MDI Visual Basic 6.0 ont changé dans Visual Basic 2005. Pour plus d'informations, consultez Objet MDIForm pour les utilisateurs de Visual Basic 6.0.

Le comportement des applications MDI a également changé. Dans Visual Basic 6.0, une application MDI comportant un formulaire qui n'est pas un enfant MDI ne se termine qu'à la fermeture du formulaire, même si le parent MDI est fermé. Dans Visual Basic 2005, l'application se termine à la fermeture du formulaire de démarrage, indépendamment de la présence ou non de formulaires non-MDI dans l'application.

Voir aussi

Concepts

Objet MDIForm pour les utilisateurs de Visual Basic 6.0
Windows Forms pour les utilisateurs de Visual Basic 6.0

Autres ressources

Applications d'interface multidocument (MDI, Multiple Document Interface)
Aide pour les utilisateurs de Visual Basic 6.0

Modifications de l'impression pour les utilisateurs de Visual Basic 6.0

Visual Basic 2005 offre une prise en charge améliorée de l'impression par l'intermédiaire de classes qui contrôlent l'impression des documents, permettent aux utilisateurs de modifier les paramètres d'impression et de choisir des imprimantes et autorisent l'aperçu avant impression.

Différences conceptuelles

Objet Printer

Dans Visual Basic 6.0, l'objet **Printer** est utilisé pour imprimer ; il prend en charge différentes méthodes graphiques telles que **Print**, **Line** et **PaintPicture** pour contrôler ce qui est imprimé.

Dans Visual Basic 2005, le composant PrintDocument remplace l'objet **Printer**. Les méthodes graphiques équivalentes sont DrawString, DrawLine et DrawImage de la classe Graphics.

Pour plus d'informations, consultez Objet imprimante pour les utilisateurs de Visual Basic 6.0.

Collection Printers

Dans Visual Basic 6.0, la collection **Printers** est utilisée pour diriger la sortie d'un objet **Printer** vers une autre imprimante. La collection **Printers** comprend la liste des imprimantes disponibles correspondant à la liste affichée dans la boîte de dialogue **Imprimer** de Windows.

Dans Visual Basic 2005, la collection **Printers** est remplacée par le contrôle PrintDialog, qui fournit une boîte de dialogue Windows standard **Imprimer**.

Pour plus d'informations, consultez Collection d'imprimantes pour les utilisateurs de Visual Basic 6.0.

Méthode PrintForm

Dans Visual Basic 6.0, la méthode **PrintForm** d'un formulaire peut être utilisée pour envoyer une image du formulaire à l'imprimante. La méthode **PrintForm** n'est plus prise en charge dans Visual Basic 2005.

Les résultats de la méthode **PrintForm** sont très variables, selon la résolution de l'écran et de celle de l'imprimante ; cette méthode n'est pas recommandée pour l'impression. Si vous avez besoin de dupliquer la fonctionnalité de la méthode **PrintForm**, vous pouvez automatiser les fonctionnalités de capture d'écran d'un outil graphique tiers pour capturer et imprimer l'image d'un formulaire.

Aperçu avant impression

Dans Visual Basic 6.0, l'implémentation de l'aperçu avant impression ne peut se faire que par l'intermédiaire de contrôles tiers. Dans Visual Basic 2005, le contrôle PrintPreviewDialog peut être utilisé pour fournir une boîte de dialogue **Aperçu avant impression** de Windows standard. Pour plus d'informations, consultez
Vue d'ensemble du contrôle PrintPreviewDialog (Windows Forms).

Mise en page

Dans Visual Basic 6.0, aucune interface utilisateur n'est proposée pour la mise en page. Dans Visual Basic 2005, le contrôle PageSetupDialog peut être utilisé pour fournir une boîte de dialogue **Mise en page** de Windows standard. Pour plus d'informations, consultez Vue d'ensemble du composant PageSetupDialog (Windows Forms).

Impression à partir d'un fichier

Dans Visual Basic 6.0, l'impression de texte à partir d'un fichier requiert une quantité importante de code. Dans Visual Basic 2005, la classe StreamReader peut être utilisée pour passer directement le contenu d'un fichier texte à un composant **PrintDocument**. Pour plus d'informations, consultez la méthode Print de la classe **PrintDocument**.

Voir aussi

Concepts

Objet imprimante pour les utilisateurs de Visual Basic 6.0
Collection d'imprimantes pour les utilisateurs de Visual Basic 6.0

Autres ressources

Prise en charge de l'impression dans les Windows Forms

Prise en charge des info-bulles pour les utilisateurs de Visual Basic 6.0

Les méthodes d'affichage des info-bulles de Visual Basic 6.0 et de Visual Basic 2005 sont complètement différentes.

Différences conceptuelles

Dans Visual Basic 6.0, la propriété **ToolTipText** d'un contrôle est utilisée pour afficher une info-bulle au moment de l'exécution.

Dans Visual Basic 2005, un seul composant **ToolTip** peut être utilisé pour contrôler les info-bulles de tous les contrôles sur un formulaire. Le composant **ToolTip** peut être ajouté dans un formulaire à partir de la **boîte à outils**. La méthode **SetToolTip** est utilisée pour définir le texte de chaque info-bulle en fonction du nom du contrôle.

Masquage d'info-bulles

Dans Visual Basic 6.0, si la propriété **ToolTipText** contient du texte, l'info-bulle s'affiche ; si elle est vide, l'info-bulle ne s'affiche pas. Pour effacer le texte de plusieurs Info-bulles, vous devez exécuter une boucle sur la collection **Controls** et définir la propriété **ToolTipText** avec la valeur d'une chaîne vide.

Dans Visual Basic 2005, vous pouvez empêcher l'affichage de l'info-bulle en passant une chaîne vide à la méthode **SetToolTip**. Vous pouvez effacer le texte de tous les info-bulles associées à un composant **ToolTip** en attribuant à la propriété **Active** la valeur **false**.

Personnalisation des info-bulles

Dans Visual Basic 6.0, il n'est pas possible de personnaliser une info-bulle sans avoir recours à des appels d'API Windows.

Dans Visual Basic 2005, plusieurs nouvelles propriétés vous permettent de personnaliser l'apparence ou le comportement d'une info-bulle, par exemple en changeant les couleurs, en définissant un délai avant l'affichage de l'info-bulle ou en créant des info-bulles multilignes.

Modifications du code des info-bulles

Les exemples de code suivants illustrent les différences entre les techniques de codage Visual Basic 6.0 et Visual Basic 2005.

Modifications du code pour afficher une info-bulle

L'exemple suivant illustre la définition d'une info-bulle d'un contrôle **Button** dans Visual Basic 6.0 et Visual Basic 2005. L'exemple de Visual Basic 2005 suppose qu'un composant ToolTip ait été ajouté au formulaire au moment du design.

```
' Visual Basic 6.0
Button1.ToolTipText = "Save changes"
```

VB

```
' Visual Basic 2005
ToolTip1.SetToolTip(Button1, "Save changes")
```

Modifications du code pour masquer une info-bulle

L'exemple suivant illustre le masquage d'une info-bulle d'un contrôle **Button** dans Visual Basic 6.0 et Visual Basic 2005. L'exemple de Visual Basic 2005 suppose qu'un composant ToolTip ait été ajouté au formulaire au moment du design.

Remarque

Le composant **ToolTip** Visual Basic 2005 possède également une propriété **Active** ; l'affectation de la valeur **false** à cette pr opriété masquera les info-bulles pour tous les contrôles associés à ce composant **ToolTip**.

```
' Visual Basic 6.0
' Hide a single ToolTip.
Button1.ToolTipText = ""
' Hide all ToolTips.
```

```
For Each Control in Me.Controls
    Control.ToolTipText = ""
Next
```

VB

```
' Visual Basic 2005
' Hide a single ToolTip.
ToolTip1.SetToolTip(Button1, "")
' Hide all ToolTips.
ToolTip1.Active = False
```

Mettre à niveau des remarques

Lorsque Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, les références à la propriété **ToolTipText** ne sont pas mises à niveau et les commentaires sont insérés dans votre code au moment du design ou au moment de l'exécution. Un composant **ToolTip** nommé `ToolTip1` est ajouté à votre formulaire. Vous devrez traiter vos info-bulles soit en définissant la propriété **ToolTip on ToolTip1** au moment du design ou en appelant la méthode **SetToolTip** au moment de l'exécution.

Si l'application Visual Basic 6.0 utilise des appels API Windows pour personnaliser des info-bulles, vous devez modifier votre code pour tirer parti des fonctionnalités de personnalisation du composant **ToolTip**.

Voir aussi

Référence

Vue d'ensemble du composant ToolTip (Windows Forms)

Autres ressources

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Numérotation des versions pour les utilisateurs de Visual Basic 6.0

Le modèle de numérotation des versions pour Visual Basic 2005 diffère de celui qui est utilisé en Visual Basic 6.0 et les méthodes de définition et de récupération des numéros de version diffèrent également.

Différences conceptuelles

Dans Visual Basic 6.0, la définition des propriétés **Major**, **Minor** et **Revision** de l'objet **App** permettent de contrôler le numéro de version d'une application. Par exemple, en affectant la valeur 1 à **Major**, la valeur 2 à **Minor** et la valeur 3 à **Revision**, le numéro de version obtenu est 1.2.0.3. Le troisième élément du numéro de version (0) ne peut pas être vu dans Visual Basic ; toutefois, il est visible lorsque vous consultez les propriétés d'un fichier compilées par Visual Basic 6.0.

Dans Visual Basic 2005, les propriétés de numéro de version sont remplacées par un attribut **AssemblyVersion** composé de quatre éléments : **Major**, **Minor**, **Build** et **Revision**. Pour l'exemple précédent, le numéro de version résultant serait 1.2.*x*.4, où *x* correspond au numéro de build. Notez que la valeur **Revision** correspond désormais au quatrième élément du numéro de version.

Définition des propriétés de version

Dans Visual Basic 6.0, les propriétés de numéro de version sont définies dans la boîte de dialogue **Propriétés du projet** ; dans Visual Basic 2005, elles sont définies dans la boîte de dialogue **Informations sur l'assembly**, disponible en cliquant sur le bouton **Informations sur l'assembly** situé dans l'onglet **Application** du **Concepteur de projets**.

Remarque
Les propriétés de numéro de version Visual Basic 6.0 sont stockées dans le fichier projet (.vbp) et peuvent également être modifiées directement à l'aide d'un éditeur de texte tel que le Bloc-notes. Les attributs d'assembly sont stockés dans le fichier AssemblyInfo.vb et peuvent également être modifiés directement dans l'éditeur de code.

Modifications du code pour la numérotation des versions

Le code suivant illustre les différences entre les techniques de codage de Visual Basic 6.0 et de Visual Basic 2005.

Modifications du code pour l'affichage du numéro de version d'une application

L'exemple suivant illustre la récupération du numéro de version d'une application et l'affichage de celui-ci dans une étiquette.

```
' Visual Basic 6.0
Label1.Caption = "Version: " & App.Major & "." & App.Minor & "." _
& App.Revision
```

VB

```
'Visual Basic 2005
Label1.Text = My.Application.Info.Version.ToString()
```

Équivalences des propriétés de numéro de version

Le tableau suivant répertorie les propriétés de numéro de version Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	**Visual Basic 2005**
Pas d'équivalent	**My.Application.AppInfo.Version.Build**
Major	**My.Application.AppInfo.Version.Major**
Minor	**My.Application.AppInfo.Version.Minor**

Revision	**My.Application.AppInfo.Version.Revision** **Remarque** Dans Visual Basic 6.0, **Revision** est le quatrième élément du numéro de version ; dans Visual Basic 2005, c'est le troisième.

Mettre à niveau des remarques

Lorsqu'une application est mise à niveau depuis Visual Basic 6.0, seules les propriétés **Major** et **Minor** sont mises à niveau ; Visual Basic 2005 assigne de nouvelles valeurs pour les propriétés **Revision** et **Build**.

Voir aussi

Concepts

Objet App pour les utilisateurs de Visual Basic 6.0

Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application

Nous vous recommandons d'utiliser les composants OpenFileDialog et SaveFileDialog pour fournir l'accès au système de fichiers ; toutefois, si vous pensez qu'il est nécessaire de créer vos propres boîtes de dialogue Fichier, Visual Basic 2005 fournit des contrôles **DirListBox**, **DriveListBox** et **FileListBox** dans le cadre de la bibliothèque Runtime Microsoft Visual Basic.

Vous pouvez ajouter un **DirListBox**, **DriveListBox** ou **FileListBox** à un projet d'application Windows à l'aide des procédures suivantes.

Remarque
Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Ajout de contrôles à votre projet

Pour ajouter une référence à la bibliothèque de compatibilité

1. Dans le menu **Projet**, cliquez sur **Ajouter une référence**.
2. Dans la boîte de dialogue **Ajouter une référence**, cliquez sur l'onglet **.NET**. Sous la liste **Nom du composant**, choisissez **Microsoft.VisualBasic.Compatibility**, puis cliquez sur **OK**.

   La référence sera ajoutée à votre projet.

Pour ajouter les contrôles à la Boîte à outils

1. Dans le menu **Affichage**, choisissez **Boîte à outils**.
2. Développez la section **Tous les Windows Forms**. Cliquez avec le bouton droit sur sa barre de titre puis cliquez sur **Choisir les éléments**.
3. Dans la boîte de dialogue **Choisir des éléments de boîte à outils**, vérifiez les composants **DirListBox**, **DriveListBox** et **FileListBox**, puis cliquez sur **OK**.

   Les contrôles seront ajoutés à la **Boîte à outils** ; ils peuvent être utilisés comme tout autre contrôle Windows Forms.

Remarque
Une fois ajoutés, les contrôles resteront dans la **Boîte à outils** jusqu'à ce que vous les supprimiez explicitement.

Voir aussi

Référence

Vue d'ensemble du composant OpenFileDialog (Windows Forms)
Vue d'ensemble du composant SaveFileDialog (Windows Forms)

Concepts

Contrôle DirListBox pour les utilisateurs de Visual Basic 6.0
Contrôle DriveListBox pour les utilisateurs de Visual Basic 6.0
Contrôle FileListBox pour les utilisateurs de Visual Basic 6.0

Débogage pour les utilisateurs de Visual Basic 6.0

Bien que le processus de débogage d'une application dans Visual Basic 2005 soit pour l'essentiel semblable à celui de Visual Basic 6.0, il existe quelques différences mineures. On compte par ailleurs plusieurs nouvelles fonctionnalités liées au débogage dans Visual Basic 2005. Pour plus d'informations, consultez Débogage de l'application Visual Basic.

Différences conceptuelles

Réglage des montres

Dans Visual Basic 6.0, les montres peuvent être réglées au moment du design, ou au moment de l'exécution, lorsque vous vous trouvez en mode arrêt. Dans Visual Basic 2005, les montres peuvent être réglées uniquement en mode arrêt.

Erreurs de compilation

Dans Visual Basic 6.0, les erreurs de compilation provoquent l'affichage d'un message d'erreur pendant le débogage.

Dans Visual Basic 2005, les erreurs de compilation sont signalées dans la fenêtre de l'éditeur de code à l'aide d'un trait de soulignement bleu. Si vous essayez de déboguer l'application avant de résoudre une erreur, un avertissement d'erreur de génération s'affiche et l'erreur apparaît dans la fenêtre **Liste des tâches**.

Exceptions runtime

Dans Visual Basic 6.0, lorsqu'une exception se produit en cours de débogage, une boîte de dialogue modale s'affiche avec un numéro d'erreur. La fermeture de la boîte de dialogue vous conduit à la ligne de code problématique dans l'**éditeur de code**.

Dans Visual Basic 2005, les exceptions provoquent l'affichage de l'**Assistant Exception** non modal dans l'**éditeur de code**. L'**Assistant Exception** affiche le type d'exception et fournit des conseils de résolution des problèmes et des actions correctives. Pour plus d'informations, consultez Assistant Exception.

Raccourcis clavier de débogage

Dans Visual Studio, certains des raccourcis clavier de débogage varient selon les langages. Vous pouvez personnaliser les paramètres en fonction de vos préférences.

Lorsque vous exécutez Visual Basic 2005 pour la première fois, vous êtes invité à choisir vos paramètres de développement. Si vous choisissez **Paramètres de Visual Basic**, les raccourcis clavier correspondent à ceux qui sont utilisés dans Visual Basic 6.0.

Remarque
Si vos raccourcis clavier ne correspondent pas aux paramètres Visual Basic 6.0, vous pouvez appliquer les **Paramètres de Visual Basic** dans l'**Assistant Importation et exportation de paramètres**, accessible depuis le menu **Outils**. Pour plus d'informations, consultez Comment : modifier les paramètres de sélection.

Voir aussi

Concepts

Débogage de l'application Visual Basic
Environnement de développement intégré pour les utilisateurs de Visual Basic 6.0

Autres ressources

Aide pour les utilisateurs de Visual Basic 6.0

Installation et déploiement pour les utilisateurs de Visual Basic 6.0

Les méthodes de déploiement des applications et des composants dans Visual Basic 2005 sont très différentes de celles de Visual Basic 6.0.

Différences conceptuelles

Dans Visual Basic 6.0, l'**Assistant Empaquetage et déploiement** est utilisé pour créer des programmes d'installation (fichiers .exe) pour distribuer et installer votre application.

Remarque
Les versions ultérieures de Visual Studio 6.0 incluent également le complément Visual Studio Installer, qui peut être utilisé p our créer des programmes d'installation Windows (fichiers .msi). Les projets créés avec Visual Studio Installer ne sont pas co mpatibles avec Visual Basic 2005.

Visual Basic 2005 fournit deux stratégies différentes pour déployer des applications Windows : publier une application à l'aide de technologies ClickOnce, ou le déployer au moyen d'un programme d'installation traditionnel à l'aide de technologies Windows Installer

Avec le déploiement ClickOnce, vous *publiez* l'application sur un emplacement centralisé (généralement un serveur Web ou un partage de fichiers), et l'utilisateur installe ou exécute l'application à partir de cet emplacement. Les applications ClickOnce peuvent être à mise à jour automatique - lorsque vous publiez une mise à jour de l'application, les utilisateurs finaux peuvent la télécharger automatiquement - garantissant ainsi que chacun utilise la même version. Pour plus d'informations, consultez Déploiement ClickOnce.

Avec le déploiement de Windows Installer, vous utilisez un projet Installation et déploiement pour empaqueter l'application dans un fichier setup.exe et distribuer ce fichier aux utilisateurs, qui utilisent le fichier setup.exe pour installer l'application. Pour plus d'informations, consultez Déploiement de Windows Installer.

Remarque
Visual Basic Express prend uniquement en charge le déploiement ClickOnce.

Composants requis et dépendances

Dans Visual Basic 6.0, les informations concernant les dépendances de fichier pour une application sont stockées dans les fichiers de dépendance (.dep). Toutes les applications nécessitent le fichier exécutable Visual Basic comme composant requis. Certaines nécessitent d'autres composants requis, tels que des bibliothèques de données. Déterminer la liste complète des dépendances et des composants requis nécessite souvent de modifier manuellement le fichier Vb6dep.ini.

Dans Visual Basic 2005, le déploiement ClickOnce et le déploiement de Windows Installer peuvent déterminer et empaqueter automatiquement toutes les dépendances pour votre application. Les composants requis tels que la bibliothèque runtime .NET Framework peuvent également être installés automatiquement avec l'un ou l'autre type de déploiement, et vous pouvez *bootstraper* (installer conjointement à votre application) n'importe quel autre composant requis. Pour plus d'informations, consultez Déploiement des composants requis (Visual Studio).

Mettre à niveau des remarques

Lorsque vous mettez à niveau une application Visual Basic 6.0 vers Visual Basic 2005, toute information de déploiement est perdue. Vous devrez implémenter un nouveau déploiement à l'aide de ClickOnce ou de Windows Installer.

Voir aussi

Concepts

Choix d'une stratégie de déploiement

Autres ressources

Déploiement d'applications et de composants

Localisation et globalisation pour les utilisateurs de Visual Basic 6.0

Visual Basic 6.0 et Visual Basic 2005 prennent tous deux en charge les applications internationales. Toutefois, les concepts et les techniques de localisation et de globalisation d'une application sont différents.

Différences conceptuelles

Ressources de localisation

Dans Visual Basic 6.0, vous pouvez créer des versions internationales d'une application en insérant toutes les informations localisables telles que les chaînes dans un fichier de ressources distinct (.res) pour chaque langue. Au moment de l'exécution, les ressources spécifiques aux paramètres régionaux sont chargées à partir du fichier de ressources en appelant les fonctions **LoadResString**, **LoadResPicture** et **LoadResData**.

Dans Visual Basic 2005, vous créez les versions internationales d'une application en modifiant la propriété **Language** d'un formulaire au moment du design. Un fichier de ressources distinct (.resx) est automatiquement créé pour chaque paramètre régional sélectionné. Vous n'avez plus besoin de charger explicitement les ressources depuis le code. Les ressources sont automatiquement chargées en fonction des paramètres régionaux de l'utilisateur. Pour plus d'informations, consultez Globalisation d'applications Windows Forms.

Modification des ressources

Les fichiers de ressources Visual Basic 6.0 peuvent être modifiés à l'aide du complément d'éditeur de ressources ou des éditeurs de ressources Visual C++.

Dans Visual Basic 2005, l'Éditeur de ressources est intégré à l'IDE comme faisant partie du **Concepteur de projets**. Pour plus d'informations, consultez Gestion des ressources de l'application.

Unicode

Dans Visual Basic 6.0, les chaînes sont représentées en interne par des caractères Unicode, mais sont affichées à l'aide de pages de code Windows. Pour la conversion de pages de code entre ANSI et DBCS, il fallait utiliser la fonction **StrConv** avec les versions binaire et Unicode des fonctions de manipulation de chaîne (par exemple, **ChrB** et **ChrW**).

Dans Visual Basic 2005, les formulaires sont totalement compatibles Unicode. La conversion entre les pages de code n'est plus nécessaire. Pour plus d'informations, consultez Codage et globalisation des applications Windows Forms.

Format de date et de devise

Dans Visual Basic 6.0, le formatage des dates et des devises nécessite une attention particulière. Les valeurs entrées sous forme de texte peuvent être interprétées de façon incorrecte lors de leur conversion en dates ou en devises dans les applications localisées.

Dans Visual Basic 2005, les dates et la monnaie sont automatiquement formatées en fonction de la culture de l'utilisateur. Vous pouvez substituer les paramètres, si nécessaire, par le biais de fonctions dans l'espace de noms **System.Globalization**. Pour plus d'informations, consultez Classes spécifiques à la culture pour les Windows Forms et les Web Forms globaux.

Voir aussi

Tâches

Comment : récupérer des ressources localisées dans Visual Basic

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Déploiement et localisation

Autres ressources

Globalisation et localisation d'applications

Espaces de noms de globalisation et de localisation dans Visual Studio

Programmation de l'API Windows pour les utilisateurs de Visual Basic 6.0

L'utilisation d'API Windows n'est pas généralement nécessaire dans Visual Basic 2005, et les techniques pour les appeler diffèrent un peu de celles qui sont utilisées dans Visual Basic 6.0.

Différences conceptuelles

Dans Visual Basic 6.0, il était souvent nécessaire d'appeler les API Windows pour apporter une fonctionnalité avancée que Visual Basic lui-même ne fournissait pas.

Visual Basic 2005 incorpore le .NET Framework. Dans la plupart des cas, les appels d'API Windows ne sont plus nécessaires car nombre des fonctionnalités de l'API Windows sont encapsulées dans le .NET Framework.

Déclaration des API Windows

Dans Visual Basic 6.0, l'instruction **Declare** est utilisée pour appeler l'API Windows dans le but d'accéder aux fonctionnalités non fournies par Visual Basic.

Dans Visual Basic 2005, vous pouvez encore utiliser l'instruction **Declare** pour appeler une API, mais vous devez avoir connaissance de certaines différences. Par exemple, certains types de données dans Visual Basic 2005 ne possèdent pas d'équivalent direct ; vous devez les marshaler pour pouvoir les passer à un appel d'API. Pour plus d'informations, consultez Procédure pas à pas : appel des API Windows.

Mettre à niveau des remarques

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, les appels aux API Windows sont mis à niveau à leur place, et ne sont pas remplacés par leurs équivalents .NET Framework. Lorsque des problèmes de type de données surviennent, des avertissements de mise à niveau sont ajoutés à votre code et au rapport de mise à niveau. Dans la plupart des cas, vous souhaiterez remplacer ces appels API par leurs équivalents .NET Framework.

Voir aussi

Référence

Declare, instruction

Concepts

Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Autres ressources

Aide pour les utilisateurs de Visual Basic 6.0

Accès au Registre pour les utilisateurs de Visual Basic 6.0

Les techniques d'accès au Registre dans Visual Basic 2005 diffèrent de celles qui sont utilisées dans Visual Basic 6.0.

Différences conceptuelles

Dans Visual Basic 6.0, vous pouvez accéder au Registre à l'aide des fonctions **GetSetting** ou **SaveSetting**, qui permettent d'accéder à un ensemble limité de clés du Registre. Pour accéder à d'autres clés du Registre, les appels à l'API Windows sont requis.

Visual Basic 2005 fournit l'objet **My.Computer.Registry** qui donne accès aux clés du Registre. Pour plus d'informations, consultez Lecture et écriture dans le Registre à l'aide de My.

De plus, l'accès complet au Registre est fourni par le biais de la classe **Registry** dans le .NET Framework.

Remarque
Dans Visual Basic 2005, les fonctions **GetSetting** et **SaveSetting** sont encore prises en charge pour un accès limité au Registre. Pour plus d'informations, consultez Liste des registres.

Accès au Registre et déploiement

Dans Visual Basic 6.0, vous pouvez marquer les fichiers pour inscription lors de l'installation en éditant manuellement le fichier Setup.lst.

Dans Visual Basic 2005, les projets d'installation et de déploiement prennent complètement en charge l'utilisation du Registre pendant l'installation. Pour plus d'informations, consultez Déploiement et Registre.

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Dépannage : manipulation du Registre

Référence

Registry Class

My.Computer.Registry, objet

Concepts

Tâches courantes relatives au Registre

Lecture et écriture dans le Registre à l'aide de My

Lecture et écriture dans le Registre à l'aide de l'espace de noms Microsoft.Win32

Sécurité et Registre

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Dans Visual Basic 6.0, de nombreuses constantes étaient définies en langage Visual Basic. Dans Visual Basic 2005, la plupart des constantes sont remplacées par des énumérations dans le .NET Framework. Dans de nombreux cas, les constantes Visual Basic 6.0 peuvent également être encore utilisées.

Les liens suivants vous conduiront à des rubriques présentant les correspondances entre les constantes de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Si une constante n'est pas répertoriée, cela signifie qu'elle n'a aucun équivalent. Par exemple, les constantes DDE (Échange dynamique de données) ne sont pas répertoriées car DDE n'est pas pris en charge dans Visual Basic 2005.

Dans cette section

Constantes Align pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Align** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Alignment pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Alignment** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Arrange pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Arrange** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes BorderStyle pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **BorderStyle** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes CallType pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **CallType** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes CheckBox pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **CheckBox** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Color pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Color** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes CompareMethod pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **CompareMethod** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Date Format pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **DateFormat** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes DayOfWeek pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **DayOfWeek** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes File Attribute pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **FileAttribute** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes IMEMode pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **IMEMode** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Key Code pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Key Code** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Mouse Pointer pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Mouse Pointer** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes MsgBox Result pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **MsgBox Result** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes MsgBox Style pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **MsgBox Style** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes MultiSelect pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **MultiSelect** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes OLEDropEffect pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **OLEDropEffect** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes ScrollBar pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **ScrollBar** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes StartUpPosition pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **StartUpPosition** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes StrConv pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **StrConv** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Style pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Style** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes System Color pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **System Color** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes TriState pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **TriState** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes Variant Type pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Variant Type** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes VarType pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **VarType** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes VBA pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes Visual Basic pour applications de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes WeekOfYear pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **WeekOfYear** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes WindowState pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **WindowState** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Constantes WindowStyle pour les utilisateurs de Visual Basic 6.0

Répertorie les constantes **Window Style** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Rubriques connexes

Constantes et énumérations en Visual Basic

Introduit des énumérations Visual Basic 2005.

Contrôles Windows Forms pour les utilisateurs de Visual Basic 6.0

Fournit des liens vers les rubriques expliquant les différences entre les contrôles et les objets Windows Forms et Visual Basic 6.0.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Décrit les modifications apportées au langage Visual Basic.

Constantes Align pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **Align** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
VbAlignNone (0)	None
VbAlignTop (1)	Top
VbAlignBottom (2)	Bottom
VbAlignLeft (3)	Left
VbAlignRight (4)	Right

Voir aussi

Concepts

Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes Alignment pour les utilisateurs de Visual Basic 6.0

Les tableaux suivants répertorient les constantes **Alignment** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Constantes Alignment CheckBox et OptionButton

Visual Basic 6.0	Visual Basic 2005
VbLeftJustify (0)	MiddleLeft
VbRightJustify (1)	MiddleRight

Constantes Alignment TextBox

Visual Basic 6.0	Visual Basic 2005
VbLeftJustify (0)	Left
VbRightJustify (1)	Right
VbCenter (2)	Center

Constantes Alignment Label

Visual Basic 6.0	Visual Basic 2005
VbLeftJustify (0)	TopLeft
VbRightJustify (1)	TopRight
VbCenter (2)	TopCenter

Voir aussi

Concepts

Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0
Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0
Contrôle TextBox pour les utilisateurs de Visual Basic 6.0
Contrôle Label pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0
Constantes et énumérations en Visual Basic

Constantes Arrange pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **Arrange** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
VbCascade (0)	Cascade
VbTileHorizontal (1)	TileHorizontal
VbTileVertical (2)	TileVertical
VbArrangeIcons (3)	ArrangeIcons

Voir aussi

Concepts

Objet MDIForm pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes BorderStyle pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **BorderStyle** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
VbBSNone (0)	None
VbFixedSingle (1)	FixedSingle
VbSizable (2)	Sizable
VbFixedDialog (3)	FixedDialog
VbFixedToolWindow (4)	FixedToolWindow
VbSizableToolWindow (5)	SizableToolWindow

Voir aussi

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes CallType pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **CallType** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, vous pouvez encore utiliser les constantes **CallType** de Visual Basic 6.0 dans le co de Visual Basic 2005. Pour plus d'informations, consultez CallType, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbMethod	**CallType.Method**
vbGet	**CallType.Get**
vbSet	**CallType.Set**
vbLet	**CallType.Let**

Voir aussi

Référence

CallByName, fonction

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes CheckBox pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **CheckBox** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbUnchecked (0)	Unchecked
vbChecked (1)	Checked
vbGrayed (2)	Indeterminate

Voir aussi
Concepts
Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0
Autres ressources
Équivalents des constantes pour les utilisateurs de Visual Basic 6.0
Constantes et énumérations en Visual Basic

Constantes Color pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **Color** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbBlack	Black
vbRed	Red
vbGreen	Lime
vbYellow	Yellow
vbBlue	Blue
vbMagenta	Magenta
vbCyan	Cyan
vbWhite	White

Voir aussi

Référence

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

Constantes System Color pour les utilisateurs de Visual Basic 6.0

Concepts

Comportement des couleurs pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations (Visual Basic)

Constantes CompareMethod pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **Compare** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, les constantes **Compare** de Visual Basic 6.0 peuvent également être utilisées dans Visual Basic 2005. Pour plus d'informations, consultez CompareMethod, énumération.

Visual Basic 6.0	**Équivalent Visual Basic 2005**
vbBinaryCompare	**CompareMethod.Binary**
vbTextCompare	**CompareMethod.Text**
vbDatabaseCompare	Pas d'équivalent

Voir aussi

Référence

CompareMethod, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes Date Format pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **Date Format** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, vous pouvez encore utiliser les constantes **Date Format** de Visual Basic 6.0 dans l e code Visual Basic 2005. Pour plus d'informations, consultez DateFormat, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbGeneralDate	**DateFormat.GeneralDate**
vbLongDate	**DateFormat.LongDate**
vbShortDate	**DateFormat.ShortDate**
vbLongTime	**DateFormat.LongTime**
vbShortTime	**DateFormat.ShortTime**

Voir aussi

Référence

DateFormat, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes DayOfWeek pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **DayOfWeek** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, les constantes **DayOfWeek** de Visual Basic 6.0 peuvent être utilisées dans Visual Basic 2005. Pour plus d'informations, consultez FirstDayOfWeek Enumeration.

Visual Basic 6.0	Équivalent Visual Basic 2005
VbUseSystemDayOfWeek	**FirstDayOfWeek.System**
vbSunday	**FirstDayOfWeek.Sunday**
vbMonday	**FirstDayOfWeek.Monday**
vbTuesday	**FirstDayOfWeek.Tuesday**
vbWednesday	**FirstDayOfWeek.Wednesday**
vbThursday	**FirstDayOfWeek.Thursday**
vbFriday	**FirstDayOfWeek.Friday**
vbSaturday	**FirstDayOfWeek.Saturday**

Voir aussi

Référence

FirstDayOfWeek Enumeration

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes File Attribute pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **File Attribute** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour la compatibilité descendante, vous pouvez encore utiliser les constantes **File Attribute** de Visual Basic 6.0 dans votre c ode Visual Basic 2005. Pour plus d'informations, consultez FileAttribute, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbNormal	**FileAttribute.Normal**
vbReadOnly	**FileAttribute.ReadOnly**
vbHidden	**FileAttribute.Hidden**
vbSystem	**FileAttribute.System**
vbVolume	**FileAttribute.Volume**
vbDirectory	**FileAttribute.Directory**
vbArchive	**FileAttribute.Archive**
vbAlias	Pas d'équivalent

Voir aussi

Référence

FileAttribute, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes IMEMode pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **IMEMode** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbIMEAlphaDbl (7)	AlphaFull
vbIMEAlphaSng (8)	Alpha
vbIMEDisable (3)	Disable
vbIMEHiragana (4)	Hiragana
vbIMEKatakanaDbl (5)	Katakana
vbIMEKatakanaSng (6)	KatakanaHalf
vbIMEModeAlpha (8)	**Alpha**
vbIMEModeAlphaFull (7)	**AlphaFull**
vbIMEModeDisable (3)	**Disable**
vbIMEModeHangul (10)	Hangul
vbIMEModeHangulFull (9)	HangulFull
vbIMEModeHiragana (4)	**Hiragana**
vbIMEModeKatakana (5)	**Katakana**
vbIMEModeKatakanaHalf (6)	**KatakanaHalf**
vbIMEModeNoControl (0)	NoControl
vbIMEModeOff (2)	Off
vbIMEModeOn (1)	On
vbIMENoOp (0)	**NoControl**
vbIMEOff (2)	**Off**
vbIMEOn (1)	**On**

Voir aussi

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes Key Code pour les utilisateurs de Visual Basic 6.0

Les tableaux suivants répertorient les constantes **Key Code** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Constantes Key Code

Visual Basic 6.0	Équivalent Visual Basic 2005
vbKeyLButton (1)	LButton
vbKeyRButton (2)	RButton
vbKeyCancel (3)	Cancel
vbKeyMButton (4)	MButton
vbKeyBack (8)	Back
vbKeyTab (9)	Tab
vbKeyClear (12)	Clear
vbKeyReturn (13)	Return
vbKeyShift (16)	ShiftKey
vbKeyControl (17)	ControlKey
vbKeyMenu (18)	Menu
vbKeyPause (19)	Pause
vbKeyCapital (20)	Capital
vbKeyEscape (27)	Escape
vbKeySpace (32)	Space
vbKeyPageUp (33)	PageUp
vbKeyPageDown (34)	PageDown
vbKeyEnd (35)	End
vbKeyHome (36)	Home
vbKeyLeft (37)	Left
vbKeyUp (38)	Up
vbKeyRight (39)	Right
vbKeyDown (40)	Down

vbKeySelect (41)	Select
vbKeyPrint (42)	Print
vbKeyExecute (43)	Execute
vbKeySnapshot (44)	Snapshot
vbKeyInsert (45)	Insert
vbKeyDelete (46)	Delete
vbKeyHelp (47)	Help
vbKeyNumlock (144)	Numlock
vbKeyScrollLock (145)	Scroll

Constantes Key Code alphabétiques

Visual Basic 6.0	**Équivalent Visual Basic 2005**
VbKeyA (65)	A
VbKeyB (66)	B
VbKeyC (67)	C
VbKeyD (68)	D
VbKeyE (69)	E
VbKeyF (70)	F
VbKeyG (71)	G
VbKeyH (72)	H
VbKeyI (73)	I
VbKeyJ (74)	J
VbKeyK (75)	K
VbKeyL(76)	L
vbKeyM (77)	M
VbKeyN (78)	N
VbKeyO (79)	O
VbKeyP(80)	P

VbKeyQ (81)	Q
VbKeyR (82)	R
VbKeyS (83)	S
VbKeyT (84)	T
VbKeyU (85)	U
VbKeyV (86)	V
VbKeyW (87)	W
VbKeyX (88)	X
VbKeyY (89)	Y
VbKeyZ (90)	Z

Constantes Key numériques

Visual Basic 6.0	**Équivalent Visual Basic 2005**
vbKey0 (48)	D0
vbKey1 (49)	D1
vbKey2 (50)	D2
vbKey3 (51)	D3
vbKey4 (52)	D4
vbKey5 (53)	D5
vbKey6 (54)	D6
vbKey7 (55)	D7
vbKey8 (56)	D8
vbKey9 (57)	D9

Constantes Keypad Key numériques

Visual Basic 6.0	**Équivalent Visual Basic 2005**
vbKeyNumpad0 (96)	NumPad0
vbKeyNumpad1 (97)	NumPad1
vbKeyNumpad2 (98)	NumPad2
vbKeyNumpad3 (99)	NumPad3
vbKeyNumpad4 (100)	NumPad4

vbKeyNumpad5 (101)	NumPad5
vbKeyNumpad6 (102)	NumPad6
vbKeyNumpad7 (103)	NumPad7
vbKeyNumpad8 (104)	NumPad8
vbKeyNumpad9 (105)	NumPad9
VbKeyMultiply (106)	Multiply
VbKeyAdd (107)	Add
VbKeySeparator (108)	Separator
VbKeySubtract (109)	Subtract
VbKeyDecimal (110)	Decimal
VbKeyDivide (111)	Divide

Constantes Key Function

Visual Basic 6.0	**Équivalent Visual Basic 2005**
vbKeyF1 (112)	F1
vbKeyF2 (113)	F2
vbKeyF3 (114)	F3
vbKeyF4 (115)	F4
vbKeyF5 (116)	F5
vbKeyF6 (117)	F6
vbKeyF7 (118)	F7
vbKeyF8 (119)	F8
vbKeyF9 (120)	F9
vbKeyF10 (121)	F10
vbKeyF11 (122)	F11
vbKeyF12 (123)	F12
vbKeyF13 (124)	F13
vbKeyF14 (125)	F14
vbKeyF15 (126)	F15

vbKeyF16 (127)	F16

Voir aussi

Référence

Keys Enumeration

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes Mouse Pointer pour les utilisateurs de Visual Basic 6.0

Le tableau suivant illustre les constantes **Mouse Pointer** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
VbDefault (0)	Default
VbArrow (1)	Arrow
VbCrosshair (2)	Cross
VbIbeam (3)	IBeam
VbIconPointer (4)	Obsolète. Remplacé par **Default**
VbSizePointer (5)	SizeAll
VbSizeNESW (6)	SizeNESW
VbSizeNS (7)	SizeNS
VbSizeNWSE (8)	SizeNWSE
VbSizeWE (9)	SizeWE
VbUpArrow (10)	UpArrow
VbHourglass (11)	WaitCursor
VbNoDrop (12)	No
VbArrowHourglass (13)	AppStarting
VbArrowQuestion (14)	Help
VbSizeAll (15)	**SizeAll**
VbCustom (99) Impossible de définir un pointeur de souris personnalisé	Pas d'équivalent. Pour plus d'informations, consultez Impossible de définir un pointeur de souris personnalisé.

Voir aussi

Concepts

MousePointer pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes MsgBox Result pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **MsgBox Result** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, vous pouvez encore utiliser les constantes **MsgBox Result** de Visual Basic 6.0 dans le code Visual Basic 2005. Pour plus d'informations, consultez MsgBoxResult, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbOK	**MsgBoxResult.OK**
vbCancel	**MsgBoxResult.Cancel**
vbAbort	**MsgBoxResult.Abort**
vbRetry	**MsgBoxResult.Retry**
vbIgnore	**MsgBoxResult.Ignore**
vbYes	**MsgBoxResult.Yes**
vbNo	**MsgBoxResult.No**

Voir aussi

Référence

MsgBoxResult, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes MsgBox Style pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **MsgBox Style** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, vous pouvez encore utiliser les constantes **MsgBox Style** de Visual Basic 6.0 dans le code Visual Basic 2005. Pour plus d'informations, consultez MsgBoxStyle, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbOKOnly	**MsgBoxStyle.OKOnly**
vbOKCancel	**MsgBoxStyle.OKCancel**
vbAbortRetryIgnore	**MsgBoxStyle.AbortRetryIgnore**
vbYesNoCancel	**MsgBoxStyle.YesNoCancel**
vbYesNo	**MsgBoxStyle.YesNo**
vbRetryCancel	**MsgBoxStyle.RetryCancel**
vbCritical	**MsgBoxStyle.Critical**
vbQuestion	**MsgBoxStyle.Question**
vbExclamation	**MsgBoxStyle.Exclamation**
vbInformation	**MsgBoxStyle.Information**
vbDefaultButton1	**MsgBoxStyle.DefaultButton1**
vbDefaultButton2	**MsgBoxStyle.DefaultButton2**
vbDefaultButton3	**MsgBoxStyle.DefaultButton3**
vbDefaultButton4	Pas d'équivalent
vbApplicationModal	**MsgBoxStyle.ApplicationModal**
vbSystemModal	**MsgBoxStyle.SystemModal**
vbMsgBoxHelpButton	**MsgBoxStyle.MsgBoxHelp**
vbMsgBoxRight	**MsgBoxStyle.MsgBoxRight**
vbMsgBoxRtlReading	**MsgBoxStyle.MsgBoxRtlReading**
vbMsgBoxSetForeground	**MsgBoxStyle.MsgBoxSetForeground**

Voir aussi

Référence

MsgBoxStyle, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes MultiSelect pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **MultiSelect** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbMultiSelectNone (0)	One
vbMultiSelectSimple (1)	MultiSimple
vbvbMultiSelectExtended (2)	MultiExtended

Voir aussi

Concepts

Contrôle ListBox pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes OLEDropEffect pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **OLEDropEffect** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
VbDropEffectNone (0)	None
VbDropEffectCopy (1)	Copy
VbDropEffectMove (2)	Move
VbDropEffectScroll (&H80000000)	Scroll

Voir aussi

Concepts

Fonction de glisser-déplacer pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes ScrollBar pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **ScrollBar** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbSBNone (0)	None
vbHorizontal (1)	Horizontal
vbVertical (2)	Vertical
vbBoth (3)	Both

Voir aussi

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Contrôle PictureBox pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes StartUpPosition pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **StartUpPosition** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbStartUpManual (0)	Manual
vbStartUpOwner (1)	CenterParent
vbStartUpScreen (2)	CenterScreen
vbStartUpWindowsDefault (3)	WindowsDefaultLocation

Voir aussi

Concepts

Objet Form pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes StrConv pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **StrConv** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour la compatibilité descendante, vous pouvez encore utiliser les constantes **StrConv** de Visual Basic 6.0 dans le code Visual Basic 2005. Pour plus d'informations, consultez VbStrConv, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbUpperCase	**VbStrConv.UpperCase**
vbLowerCase	**VbStrConv.LowerCase**
vbProperCase	**VbStrConv.ProperCase**
vbWide	**VbStrConv.Wide**
vbNarrow	**VbStrConv.Narrow**
vbKatakana	**VbStrConv.Katakana**
vbHiragana	**VbStrConv.Hiragana**
vbUnicode	Pas de constante équivalente. Vous pouvez obtenir la même fonctionnalité à l'aide de la méthode Convert.
vbFromUnicode	Pas d'équivalent

Voir aussi

Référence

VbStrConv, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes Style pour les utilisateurs de Visual Basic 6.0

Les tables suivantes répertorient les constantes **Style** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Constantes Style ComboBox

Visual Basic 6.0	Équivalent Visual Basic 2005
vbComboDropdown (0)	DropDown
vbComboSimple (1)	Simple
vbComboDropdownList (2)	DropDownList

Constantes Style CheckBox et OptionButton

Visual Basic 6.0	Équivalent Visual Basic 2005
VbButtonStandard (0)	Normal
VbButtonGraphical (1)	Button

Remarque

Dans Visual Basic 6.0, les constantes **Style** sont définies également pour **CommandButton** et les contrôles **ListBox**. Il n'existe pas d'équivalents dans Visual Basic 2005. Pour plus d'informations, consultez Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau ou Contrôle ListBox pour les utilisateurs de Visual Basic 6.0.

Voir aussi

Concepts

Contrôle ComboBox pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes System Color pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **System Color** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vb3DDKShadow	ControlDarkDark
vb3DFace	Control
vb3DHighlight	ControlLightLight
vb3DLight	ControlLight
vb3DShadow	ControlDark
vbActiveBorder	ActiveBorder
vbActiveTitleBar	ActiveCaption
vbActiveTitleBarText	ActiveCaptionText
vbApplicationWorkspace	AppWorkspace
vbButtonFace	**Control**
vbButtonShadow	**ControlDark**
vbButtonText	ControlText
vbDesktop	Desktop
vbGrayText	GrayText
vbHighlight	Highlight
vbHighlightText	HighlightText
vbInactiveBorder	InactiveBorder
vbInactiveCaptionText	InactiveCaptionText
vbInactiveTitleBar	InactiveCaption
vbInactiveTitleBarText	**InactiveCaptionText**
vbInfoBackground	Info
vbInfoText	InfoText
vbMenuBar	Menu

vbMenuText	MenuText
vbScrollBars	Scrollbar
vbTitleBarText	**ActiveCaptionText**
vbWindowBackground	Window
vbWindowFrame	WindowFrame
vbWindowText	WindowText

Voir aussi

Référence

Gestion des couleurs pour les utilisateurs de Visual Basic 6.0

Constantes Color pour les utilisateurs de Visual Basic 6.0

Concepts

Comportement des couleurs pour les utilisateurs de Visual Basic 6.0

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes TriState pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **TriState** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, les constantes **TriState** de Visual Basic 6.0 peuvent toujours être utilisées dans Visual Basic 2005. Pour plus d'informations, consultez Tristate, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbTrue	**TriState.True**
vbFalse	**TriState.False**
vbUseDefault	**TriState.UseDefault**

Voir aussi

Référence

Tristate, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes Variant Type pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **VarType** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbVEmpty	**VariantType.Empty**
vbVNull	**VariantType.Null**
vbVInteger	**VariantType.Short**
vbVLong	**VariantType.Integer**
vbVSingle	**VariantType.Single**
vbVDouble	**VariantType.Double**
vbVCurrency	**VariantType.Decimal**
vbVDate	**VariantType.Date**
vbVString	**VariantType.String**

Voir aussi

Référence

VariantType, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations (Visual Basic)

Constantes VarType pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **VarType** de Visual Basic 6.0 et leurs équivalents Visual Basic 2005 dans l'énumération VariantType.

Remarque

Pour assurer la compatibilité descendante, les constantes **VarType** de Visual Basic 6.0 peuvent toujours être utilisées dans Visual Basic 2005. Pour plus d'informations, consultez VariantType, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbEmpty	**VariantType.Empty**
vbNull	**VariantType.Null**
vbInteger	**VariantType.Short**
vbLong	**VariantType.Integer**
vbSingle	**VariantType.Single**
vbDouble	**VariantType.Double**
vbCurrency	**VariantType.Decimal**
vbDate	**VariantType.Date**
vbString	**VariantType.String**
vbObject	**VariantType.Object**
vbError	**VariantType.Error**
vbBoolean	**VariantType.Boolean**
vbVariant	**VariantType.Variant**
vbDataObject	**VariantType.DataObject**
vbDecimal	**VariantType.Decimal**
vbByte	**VariantType.Byte**
vbUserDefinedType	**VariantType.UserDefinedType**
vbArray	**VariantType.Array**

Voir aussi

Référence

VariantType, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes VBA pour les utilisateurs de Visual Basic 6.0

Le tableau suivant montre les constantes Microsoft Visual Basic pour Applications (VBA) pour Visual Basic 6.0 et leurs équivalents Visual Basic 2005.

Remarque
Pour assurer la compatibilité descendante, les constantes de Visual Basic 6.0 VBA peuvent également être utilisées dans Visual Basic 2005.

Visual Basic 6.0	**Équivalent Visual Basic 2005**
vbBack	**ControlChars.Back**
vbCr	**ControlChars.Cr**
vbCrLf	**ControlChars.CrLf**
vbFormFeed	**ControlChars.FormFeed**
vbLf	**ControlChars.Lf**
vbNewLine	**ControlChars.NewLine**
vbNullChar	**ControlChars.NullChar**
vbNullString	Mot clé **Nothing**
vbObjectError	**vbObjectError**
vbTab	**ControlChars.Tab**

Voir aussi

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes WeekOfYear pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **WeekOfYear** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour la compatibilité descendante, vous pouvez encore utiliser les constantes **WeekOfYear** de Visual Basic 6.0 dans Visual Basic 2005. Pour plus d'informations, consultez FirstWeekOfYear, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbUseSystem	**FirstWeekOfYear.System**
VbFirstJan1	**FirstWeekOfYear.Jan1**
vbFirstFourDays	**FirstWeekOfYear.FirstFourDays**
vbFirstFullWeek	**FirstWeekOfYear.FirstFullWeek**

Voir aussi

Référence

FirstWeekOfYear, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Constantes WindowState pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **WindowState** de Visual Basic 6.0 avec leurs valeurs et leurs équivalents dans Visual Basic 2005.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbNormal (0)	Normal
vbMinimized (1)	Minimized
vbMaximized (2)	Maximized

Voir aussi
Concepts
Objet Form pour les utilisateurs de Visual Basic 6.0
Autres ressources
Équivalents des constantes pour les utilisateurs de Visual Basic 6.0
Constantes et énumérations en Visual Basic

Constantes WindowStyle pour les utilisateurs de Visual Basic 6.0

Le tableau suivant répertorie les constantes **WindowStyle** de Visual Basic 6.0 et leurs équivalents dans Visual Basic 2005.

Remarque
Pour la compatibilité descendante, vous pouvez encore utiliser les constantes **WindowStyle** de Visual Basic 6.0 dans Visual Basic 2005. Pour plus d'informations, consultez AppWinStyle, énumération.

Visual Basic 6.0	Équivalent Visual Basic 2005
vbHide	**AppWinStyle.Hide**
vbNormalFocus	**AppWinStyle.NormalFocus**
vbMinimizedFocus	**AppWinStyle.MinimizedFocus**
vbMaximizedFocus	**AppWinStyle.MaximizedFocus**
vbNormalNoFocus	**AppWinStyle.NormalNoFocus**
vbMinimizedNoFocus	**AppWinStyle.MinimizedNoFocus**

Voir aussi

Référence

AppWinStyle, énumération

Autres ressources

Équivalents des constantes pour les utilisateurs de Visual Basic 6.0

Constantes et énumérations en Visual Basic

Bibliothèques d'objets et espaces de noms pour les utilisateurs de Visual Basic 6.0

Les espaces de noms peuvent tout d'abord sembler être un nouveau concept de Visual Basic 2005. Ils sont toutefois conceptuellement semblables aux bibliothèques d'objets de Visual Basic 6.0.

Dans Visual Basic 6.0, plusieurs bibliothèques contiennent des objets qui sont utilisés pour générer une application. Par exemple, la bibliothèque Visual Basic contient les objets formulaires et les objets contrôle intrinsèques qui constituent la base d'une application Windows. Vous pouviez visualiser les objets d'une bibliothèque donnée en l'affichant dans l'**Explorateur d'objets**.

Dans Visual Basic 2005, les objets sont contenus dans des assemblys qui appartiennent à la bibliothèque de classes .NET Framework. Chaque assembly représente un espace de noms. Par exemple, les objets formulaire et contrôle sont contenus dans l'espace de noms **System.Windows.Forms**. Comme dans Visual Basic 6.0, les espaces de noms peuvent être examinés dans l'**Explorateur d'objets**.

Conseil
L'espace de noms **Microsoft.VisualBasic**, qui contient de nombreux objets et constantes vous permettant de continuer à uti liser la syntaxe Visual Basic 6.0 dans Visual Basic 2005 devra particulièrement intéresser les utilisateurs de Visual Basic 6.0

Voir aussi

Concepts

Introduction à la bibliothèque de classes .NET Framework dans Visual Studio

Bibliothèques Runtime pour les utilisateurs de Visual Basic 6.0

Les applications créées avec Visual Basic 6.0 ou avec Visual Basic 2005 nécessitent qu'une bibliothèque Runtime soit installée sur l'ordinateur de l'utilisateur.

Différences conceptuelles

Pour les applications créées avec Visual Basic 6.0, le fichier exécutable Visual Basic (Msvbvm60.dll) doit être installé sur un ordinateur client afin de pouvoir exécuter l'application. Ce fichier était généralement inclus dans les conditionnements d'installation, afin de pouvoir être installé, si nécessaire, en même temps que l'application.

Dans Visual Basic 2005, les applications nécessitent le common language runtime .NET Framework, un environnement d'exécution qui gère l'exécution de code sur un ordinateur client. Les fichiers Common Language Runtime peuvent être préinstallés sur un ordinateur cible à l'aide du mode de déploiement ClickOnce ou Windows Installer. Pour plus d'informations, consultez Déploiement des composants requis (Visual Studio).

Voir aussi

Concepts

Installation et déploiement pour les utilisateurs de Visual Basic 6.0

Autres ressources

Déploiement d'applications et de composants

Redistribution du .NET Framework

Mise à niveau d'applications dans Visual Basic

Dans les sections suivantes, vous pouvez apprendre comment mettre à niveau des programmes créés avec Visual Basic 6.0 vers Visual Basic 2005.

Dans cette section

Nouveautés de la mise à niveau

Nouvelles fonctionnalités relatives à la mise à niveau de vos projets Visual Basic 6.0.

Mise à niveau de Visual Basic 6.0

Conversion des applications vers la nouvelle version de Visual Basic.

Référence de mise à niveau Visual Basic 6.0

Messages de diagnostic pour l'outil qui met à niveau des projets vers Visual Basic 2005.

Rubriques connexes

Aide pour les utilisateurs de Visual Basic 6.0

Ensemble des rubriques qui comparent des objets, des concepts et des techniques Visual Basic 6.0 avec leurs équivalents Visual Basic 2005.

Introduction aux solutions, projets et éléments

Décrit les concepts principaux sous-jacents du développement d'applications dans l'environnement de développement intégré (IDE, Integrated Development Environment). Cette rubrique fournit des informations générales sur la gestion de projets mis à niveau et l'exécution de tâches relatives aux projets.

Support technique et accessibilité

Fournit des informations relatives à l'installation, aux éditions, au support technique et à la personnalisation de l'IDE.

Nouveautés de la mise à niveau

Visual Basic 2005 inclut plusieurs nouvelles fonctionnalités pour mettre à niveau des applications et le code Visual Basic 6.0 et vers Visual Basic 2005.

Nouveautés de Visual Basic 2005

Fonctionnalité	Description
Prise en charge des mises à niveau MTS et COM+	Vous pouvez désormais mettre à niveau des projets Visual Basic 6.0 qui utilisaient des composants Microsoft Transaction Server (MTS) ou COM+.
Prise en charge des mises à niveau de contrôle ActiveX	Vous pouvez désormais mettre à niveau de nombreux contrôles ActiveX Visual Basic 6.0 vers leurs équivalents dans Visual Basic 2005. La prise en charge est incluse pour ces contrôles : WebBrowser, ToolBar, ImageList, TreeView, ListView, RichTextBox, ProgressBar, StatusBar, CommonDialog et MaskedTextBox.
Améliorations apportées aux mises à niveau du code	Vous trouverez plusieurs améliorations apportées aux mises à niveau du code dans Visual Basic 2005, y compris : • Mettez à niveau le code qui utilise des clés basées sur des chaînes dans les collections de contrôle de formulaires. • Mettez à niveau le code qui modifie la valeur de clé dans l'événement KeyPress. Il existe d'autres améliorations : • Mettez à niveau l'objet **App** de Visual Basic 6.0 vers le nouvel objet **My.Application**. • Mettez à niveau le paramètre *UnloadMode* d'événement **QueryUnload** vers l'argument *ClosingReason* de l'événement **FormClosing**. • Mettez à niveau des formulaires pour utiliser le nouveau format Type partiel. • Mettez à niveau des instances par défaut de Visual Basic 6.0 pour utiliser la prise en charge des instances par défaut dans Visual Basic 2005. • Mettez à niveau la collection **Forms** vers la collection System.Windows.Forms de Visual Basic 2005. • Mettez à niveau l'objet **Clipboard** de Visual Basic 6.0 vers le nouvel objet **My.Computer.Clipboard**. • Mettez à niveau les méthodes **LoadResString**, **LoadResData** et **LoadResPicture** vers le nouvel objet **My.Resources**.

Voir aussi

Concepts

Nouveautés du langage Visual Basic

Mise à niveau de Visual Basic 6.0

Visual Basic 2005 surpasse les versions antérieures de Visual Basic, avec un nouvel environnement de développement, un langage de programmation mis à jour et un nouveau package de formulaires facilitant le développement que jamais. Le passage de Visual Basic 6.0 à Visual Basic 2005 demande moins d'efforts que vous pouvez ne le penser : les améliorations apportées à l'environnement de développement intégré et le système d'aide facilitent la tâche de mise à niveau réussie de votre code en mappant vos connaissances de Visual Basic 6.0 aux équivalents de Visual Basic 2005.

Au fur et à mesure que vous vous familiariserez avec Visual Basic 2005, il est probable que vous souhaiterez mettre à niveau certains de vos projets Visual Basic 6.0. Bien qu'il existe des différences entre les deux versions, les outils fournis peuvent vous aider à mettre à niveau avec succès vos projets.

La documentation suivante vous aidera à mettre à niveau vos applications et à vous familiariser rapidement avec les modifications apportées à Visual Basic 2005.

Dans cette section

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Décrit comment mettre à niveau vos applications vers Visual Basic 2005.

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Décrit les points de compatibilité entre les deux versions de Visual Basic.

Procédure pas à pas : mise à niveau d'une application Visual Basic 6.0 vers la version actuelle de Visual Basic

Compare un exemple d'application Visual Basic 6.0 à un équivalent dans Visual Basic 2005.

Rubriques connexes

Nouveautés pour les utilisateurs de Visual Basic 6.0

Décrit les nouvelles fonctionnalités de Visual Basic 2005.

Aide pour les utilisateurs de Visual Basic 6.0

Ensemble des rubriques qui comparent des objets, des concepts et des techniques Visual Basic 6.0 avec leurs équivalents Visual Basic 2005.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Décrit les modifications apportées à Visual Basic.

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Avec Visual Studio 2005, vous pouvez mettre à niveau des applications créées avec Visual Basic 6.0 et ainsi de continuer leur développement en bénéficiant des avantages du .NET Framework. Lorsque vous ouvrez pour la première fois un fichier projet Visual Basic 6.0 (.vbp), l'Assistant Mise à niveau apparaît. Vous pouvez également mettre à niveau des projets en dehors de l'environnement de développement avec l'outil de ligne de commande fourni.

Les outils de mise à niveau modifient le code contenu dans votre projet pour le mettre en conformité avec la syntaxe de Visual Basic 2005 et remplacent les formulaires et les contrôles par leurs équivalents Visual Basic 2005. Du fait de ces différences entre Visual Basic 6.0 et Visual Basic 2005, certaines parties de votre projet peuvent ne pas être mappées directement ; le cas échéant, vous recevez un rapport de mise à niveau pour vous guider à travers le processus de modification de votre application.

Certaines applications Visual Basic 6.0 peuvent ne pas gagner à être mises à niveau vers Visual Basic 2005. Vous pouvez préparer ces applications que vous choisissez de mettre à niveau afin que la mise à niveau soit effectuée régulièrement.

Les rubriques suivantes vous aideront à mieux comprendre les tenants et aboutissants de la mise à niveau.

Dans cette section

Pourquoi une mise à niveau ?

Étudie les différences entre Visual Basic 6.0 et Visual Basic 2005.

Points à considérer avant d'effectuer la mise à niveau

Présente les restrictions à la mise à niveau et les points à prendre en considération avant de l'effectuer.

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Décrit comment s'assurer qu'une application est prête pour la mise à niveau.

Recommandations de mise à niveau relatives au langage

Fournit quelques conseils que vous pouvez suivre lorsque vous préparez une application Visual Basic 6.0 en vue de sa mise à niveau ou lorsque vous développez des applications Visual Basic 6.0 destinées à être mises à niveau ultérieurement.

Bibliothèque de compatibilité Visual Basic 6.0

Présente un mécanisme fourni pour la compatibilité descendante.

Assistant Mise à niveau de Visual Basic

Présente une vue d'ensemble de la mise à niveau d'un projet Visual Basic 6.0 vers Visual Basic 2005 à l'aide de l'Assistant Mise à niveau.

Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic

Explique comment utiliser l'Assistant Mise à niveau.

Comment : mettre à niveau du code Visual Basic 6.0 à l'aide de la boîte de dialogue Mettre à niveau le code Visual Basic 6

Explique comment mettre à niveau du code Visual Basic 6.0 vers Visual Basic 2005.

Mettre à niveau le code Visual Basic 6, boîte de dialogue

Décrit l'interface utilisateur pour la mise à niveau de code Visual Basic 6.0 vers Visual Basic 2005.

Mise à niveau de projets WebClass vers Visual Basic 2005

Explique comment mettre à niveau des projets WebClass (également appelés applications IIS) vers des projets d'application Web ASP.NET.

Fin du processus de mise à niveau

Décrit les opérations à effectuer après que l'Assistant Mise à niveau a fini de s'exécuter.

Rapport de mise à niveau Visual Basic

Décrit le format et l'objectif du rapport de mise à niveau.

Comment : afficher le rapport de mise à niveau

Explique comment ouvrir un rapport de mise à niveau.

Comment : mettre à niveau un projet à partir de la ligne de commande

Explique comment utiliser l'outil de mise à niveau de la ligne de commande.

Dépannage des applications mises à niveau à partir de Visual Basic 6.0

Répertorie certains problèmes connus non détectés par les outils de mise à niveau et explique comment y remédier.

Rubriques connexes

Nouveautés de Visual Basic

Décrit les nouvelles fonctionnalités de Visual Basic 2005.

Aide pour les utilisateurs de Visual Basic 6.0

Répertorie les différences entre Visual Basic 6.0 et Visual Basic 2005.

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Présente les points de compatibilité entre les deux versions de Visual Basic.

Pourquoi une mise à niveau ?

Les concepteurs de Visual Basic 2005 disposaient de deux options : soit réutiliser la base de code existante pour la faire fonctionner par-dessus le .NET Framework, soit repartir de zéro et exploiter toutes les possibilités du .NET Framework. La bonne option consistait à repartir de zéro. Le résultat est un produit qui :

- Fournit les fonctionnalités les plus demandées par les clients (par exemple, l'héritage et le threading).
- Fournit l'accès complet et libre au .NET Framework.
- Garantit que Visual Basic passe à la génération suivante d'applications Web.

Ainsi, nombre de nouvelles fonctionnalités proposées dans les Windows Forms (le nouveau package de formulaires) auraient pu être ajoutées à la base de code existante sous forme de nouveaux contrôles ou de propriétés supplémentaires. Toutefois, les développeurs ne profiteraient pas des possibilités de toutes les autres fonctionnalités importantes inhérentes à Windows Forms, tels que la sécurité et l'héritage visuel.

L'un des buts majeurs de Visual Basic 2005 était garantir que le code Visual Basic puisse interagir pleinement avec le code écrit dans d'autres langages, tels que C# ou C++. L'autre but étant de permettre aux développeurs Visual Basic d'exploiter simplement la puissance du .NET Framework, sans avoir recours aux solutions de programmation généralement requises pour faire fonctionner des API Windows. Visual Basic possède désormais les mêmes types de variables, tableaux, types définis par l'utilisateur, classes et interfaces que Visual C++ ou tout autre langage ciblant le Common Language Runtime ; toutefois, certaines fonctionnalités telles que les chaînes de longueur fixe ont dû être supprimées.

Visual Basic est à présent un véritable langage orienté objet ; certaines fonctionnalités incohérentes et peu intuitives, comme **GoSub**/**Return** et **DefInt**, ont été supprimées.

Visual Basic a ainsi retrouvé une nouvelle vigueur ; il demeure l'outil le plus productif de création des applications Windows et se trouve actuellement au premier rang des outils de création des sites Web de nouvelle génération. La seule ombre à ce tableau est que les applications Visual Basic 6.0 doivent être mises à niveau pour devenir conformes au nouveau langage et aux nouveaux formulaires, au détriment de la compatibilité avec les versions antérieures.

Voir aussi

Concepts

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Points à considérer avant d'effectuer la mise à niveau

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Points à considérer avant d'effectuer la mise à niveau

Visual Basic 2005 représente un décalage majeur avec les versions antérieures de Visual Basic. Parce qu'il a été conçu dès le départ pour tirer parti de toutes les possibilités offertes par le .NET Framework, il existe de nombreux points sur lesquels il n'y a pas de compatibilité avec les versions précédentes.

Remarque
Visual Basic 2005 et Visual Basic 6.0 peuvent être installés sur le même ordinateur et s'exécuter simultanément. Les applications écrites en Visual Basic 2005 et en Visual Basic 6.0 peuvent être installées et exécutées sur le même ordinateur. Les composants développés en Visual Basic 2005 peuvent interagir avec des composants COM développés dans des versions antérieures de Visual Basic et d'autres langages.

- Dans la plupart des cas, la mise à niveau des applications Visual Basic 6.0 vers Visual Basic 2005 est effectuée dans le but de continuer leur développement en bénéficiant du .NET Framework. Toutefois, dans certains cas, vous voudrez aussi continuer le développement avec Visual Basic 6.0 sans effectuer de mise à niveau. Les fonctionnalités non prises en charge par Visual Basic 2005 et la somme de travail qu'impliquent les problèmes d'incompatibilité et d'architecture font partie des critères de décision en faveur de la mise à niveau ou non d'une application.

Fonctionnalités non prises en charge

Les fonctionnalités suivantes de Visual Basic 6.0 ne sont plus prises en charge dans Visual Basic 2005. Pour plus d'informations, consultez Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau.

- Contrôle conteneur OLE Il n'existe aucun équivalent dans Visual Basic 2005. Les applications dépendant de ce contrôle doivent être laissées en Visual Basic 6.0.
- Échange dynamique de données (DDE, Dynamic Data Exchange) Les méthodes DDE ne sont plus prises en charge. Les applications qui dépendent de DDE doivent soit être modifiées afin d'utiliser une autre méthode de communication interapplication, soit être laissées en Visual Basic 6.0.
- Liaison de données DAO ou RDO La liaison avec une source de données DAO ou RDO n'est pas prise en charge dans Visual Basic 2005 ; les contrôles **Data** et **RemoteData** n'y ont pas d'équivalent. Il n'est possible d'accéder aux sources de données DAO et RDO que par le code. Les applications qui utilisent la liaison de données DAO ou RDO doivent soit être mises à jour soit être laissées en Visual Basic 6.0.
- Contrôles Visual Basic 5.0 Visual Basic 6.0 incluait les versions Visual Basic 5.0 des contrôles communs Windows ainsi que le contrôle de grille liée aux données. Ces contrôles sont incompatibles avec Visual Basic 2005. Les applications qui utilisent ces contrôles doivent soit être mises à jour, soit être laissées en Visual Basic 6.0.
- Applications DHTML Il n'existe aucun équivalent dans Visual Basic 2005. Toutefois, les applications DHTML peuvent interagir avec les technologies Visual Basic 2005.
- Documents ActiveX Il n'existe aucun équivalent dans Visual Basic 2005. Toutefois, les applications DHTML peuvent interagir avec les technologies Visual Basic 2005. Les documents ActiveX doivent soit être réécrits sous forme de contrôles utilisateur, soit être laissés en Visual Basic 6.0.
- Pages de propriétés Il n'existe aucun équivalent dans Visual Basic 2005. Si votre application dépend fortement de ces pages de propriétés, laissez-la en Visual Basic 6.0.

Évaluation des modifications requises

Du fait des nombreuses différences existant entre Visual Basic 6.0 et Visual Basic 2005, il sera le plus souvent nécessaire de modifier les applications, notamment les éléments incompatibles ou ne pouvant pas être mappés directement. L'importance de ces modifications dépend de plusieurs facteurs tels que le type de l'application ou les fonctionnalités et constructions de langage qu'elle utilise.

La façon la plus simple d'évaluer l'importance des modifications requises est d'exécuter l'Assistant Mise à niveau et de consulter la liste des problèmes répertoriés dans le rapport de mise à niveau. Le processus de mise à niveau crée une copie de l'application en laissant intact l'application d'origine. Si, après avoir consulté le rapport, vous renoncez à la mise à niveau, il vous suffit de supprimer le nouveau projet et de continuer le développement avec Visual Basic 6.0.

Nous avons répertorié ci-après d'autres facteurs qui peuvent peser dans votre décision de mettre ou non à niveau une

application.

- Applications de base de données monocouche La liaison avec les sources de données DAO n'étant pas prise en charge, la mise à niveau d'applications simples utilisant des contrôles directement liés à une base de données locale (créée, par exemple, dans Microsoft Access) peut impliquer des modifications importantes.
- Compléments Visual Basic Étant donné que Visual Basic 2005 utilise l'environnement de développement intégré Visual Studio, son modèle objet d'extensibilité est sensiblement différent de celui de Visual Basic 6.0. Les compléments (appelés aussi Add-In en anglais) nécessiteront par conséquent de très nombreuses modifications.
- Jeux Les applications telles que les jeux d'arcade exploitant les performances spécifiques de Visual Basic 6.0, nécessiteront des modifications ; en effet, Visual Basic 2005 n'offre pas les mêmes performances.
- Graphismes Ni les méthodes graphiques des formulaires ni les contrôles de ligne et de forme ne sont pris en charge. Les applications faisant une large utilisation de ces fonctionnalités pour créer des dessins dans les formulaires nécessiteront de très nombreuses modifications.
- Fonctionnalité de glisser-déplacer Les modèles de la fonctionnalité de glisser-déplacer sont très différents ; tout code effectuant des opérations de glisser-déplacer devra être réécrit.
- Variants Lorsqu'une application est mise à niveau, le type de données **Variant** est converti en **Object**. Si une application dépend fortement de ces données Variant, sa conversion peut introduire de subtiles différences dans son comportement. Par exemple, dans Visual Basic 6.0, l'expression `IsObject(Variant)` a la valeur **false** ; dans Visual Basic 2005, l'expression devient `IsObject(Object)` et a la valeur **true**.
- API Windows Du fait de changements liés au langage, un grand nombre d'appels aux API Windows devront être révisés ou remplacés par des fonctions du .NET Framework.

Voir aussi

Concepts

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau
Pourquoi une mise à niveau ?

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Bien que le processus de mise à niveau vers Visual Basic 2005 d'une application Visual Basic 6.0 soit en grande partie automatique, un peu de préparation vous permettra de l'optimiser.

Si vous suivez les recommandations de cette rubrique, vous pourrez réduire et, le cas échéant, supprimer les modifications exigées après la mise à niveau d'un projet vers Visual Basic 2005. Dans la plupart des cas, ces recommandations suivent les méthodes conseillées en matière de programmation ; toutefois, certaines identifient également les objets et les méthodes qui n'ont pas d'équivalent et que vous devez utiliser avec parcimonie si vous avez l'intention de mettre votre projet à niveau vers Visual Basic 2005.

Recommandations générales

Cela peut sembler évident, mais si votre projet n'est pas compilé et ne s'exécute pas dans Visual Basic 6.0, il risque de ne pas pouvoir être mis à niveau. Il est donc conseillé d'installer Visual Basic 6.0 sur l'ordinateur que vous souhaitez utiliser pour la mise à niveau. En plus de vous permettre de tester l'application, cela garantira que tous les contrôles, composants et bibliothèques de types nécessaires seront disponibles durant la mise à niveau.

Visual Basic 2005 vous permet de mettre à niveau des projets Visual Basic 6.0 ; si votre projet est écrit dans une version 1 à 5 de Visual Basic, vous devez dans un premier temps le charger dans Visual Basic 6.0 (en sélectionnant la mise à niveau des contrôles Microsoft ActiveX), le compiler et l'enregistrer avant de le mettre à niveau vers Visual Basic 2005.

Considérations relatives aux formulaires et aux contrôles

Visual Basic 2005 est doté d'un nouveau package de formulaires, Windows Forms. Windows Forms est largement compatible avec le package de formulaires de Visual Basic 6.0 ; toutefois, il existe entre ces deux packages certaines différences qui sont énumérées ci-dessous :

- Visual Basic 2005 ne prend pas en charge le contrôle conteneur OLE ; évitez donc d'utiliser ce contrôle dans vos applications Visual Basic 6.0.
- Il n'y a aucun contrôle Shape dans Visual Basic 2005. Les formes carrées et rectangulaires sont mises à niveau sous la forme d'étiquettes (Label), tandis que les ovales et les cercles ne peuvent pas être mis à niveau. Évitez de les utiliser dans vos applications.
- Il n'y a aucun contrôle Line dans Visual Basic 2005. Les lignes horizontales et verticales sont mises à niveau sous forme d'étiquettes (Label). Les lignes diagonales ne sont pas mises à niveau et vous devez par conséquent éviter de les utiliser.
- Visual Basic 2005 dispose d'un nouveau jeu de commandes graphiques qui remplacent les méthodes **Form Circle**, **CLS**, **PSet**, **Line** et **Point**. Étant donné que le nouveau modèle objet est assez différent de celui de Visual Basic 6.0, ces méthodes ne peuvent pas être mises à niveau.
- S'agissant du contrôle **Timer**, le fait d'affecter à la propriété **Interval** la valeur 0 ne désactive pas la minuterie ; l'intervalle est réinitialisé à 1. Dans vos projets Visual Basic 6.0, vous devez affecter à la propriété **Enabled** la valeur **False** plutôt que d'affecter à **Interval** la valeur 0.
- Visual Basic 2005 a deux contrôles de menu, MenuStrip et ContextMenuStrip, alors que Visual Basic 6.0 a un seul contrôle de menu qui peut être ouvert comme un contrôle **MainMenu** ou **ContextMenu**. Tous les contrôles de menu sont mis à niveau vers un composant **MenuStrip** unique contenant des MenuItem pour chaque contrôle de menu ; vous devrez recréer vos menus contextuels et supprimer les contrôles **MenuStrip** en surnombre.
- Visual Basic 2005 ne prend pas en charge l'échange dynamique de données (DDE, Dynamic Data Exchange).
- Visual Basic 2005 ne prend pas en charge la méthode **Form.PrintForm**.
- Bien que Visual Basic 2005 prenne en charge la fonctionnalité de glisser-déplacer, le modèle objet est différent de celui de Visual Basic 6.0. Les propriétés et méthodes de glisser-déplacer ne peuvent donc pas être mises à niveau.
- Visual Basic 2005 a un objet **Clipboard** amélioré (**My.Computer.Clipboard**) qui offre plus de fonctionnalités et prend en charge plus de formats de presse-papiers que l'objet **Clipboard** de Visual Basic 6.0. Toutefois, du fait des différences existant entre les modèles objet, les instructions de Presse-papiers ne peuvent pas être mises à niveau automatiquement.
- Visual Basic 2005 ne prend pas en charge la propriété **Name** pour les formulaires et les contrôles au moment de

l'exécution ; par conséquent, vous ne devez pas écrire du code qui itère la collection **Controls** recherchant un contrôle spécifique. Cette fonctionnalité est désormais disponible lorsqu'on utilise les classes .NET Framework **System.Reflection**.

Recommandations relatives aux données

Visual Basic 2005 contient une version améliorée des objets de données actifs (ADO, Active Data Objects) appelée ADO.NET, qui est optimisé pour fonctionner avec des données en applications distribuées et offre des performances supérieures à ADO dans ce contexte.

RDO et ADO peuvent encore être utilisés dans le code de Visual Basic 2005, avec quelques modifications simples. Toutefois, Visual Basic 2005 ne prend pas en charge la liaison de données DAO et RDO aux contrôles ou contrôles de données ni la connexion utilisateur RDO. Si vos applications contiennent des liaisons de données DAO ou RDO, nous vous conseillons de les laisser en Visual Basic 6.0 ou de mettre à niveau la liaison de données DAO ou RDO vers ADO avant de mettre à niveau votre projet vers Visual Basic 2005, car la liaison de données ADO est prise en charge dans les Windows Forms. Pour plus d'informations sur la mise à niveau de DAO ou RDO vers ADO dans Visual Basic 6.0, consultez l'aide de Visual Basic 6.0.

Recommandations relatives à l'architecture Web

Visual Basic 2005 introduit ASP.NET, une version améliorée d'ASP, et enrichit l'architecture existante de Web Forms qui permettent de créer des pages Web à l'aide d'un modèle d'événements de type Visual Basic. L'architecture est une architecture serveur.

Les applications Web qui répondent aux critères suivants seront les plus faciles à mettre à niveau :

- Suivez les indications de Microsoft relatives à l'architecture multicouche.
- Utilisez des pages Active Server Pages.
- Utilisez des objets Visual Basic 6.0 ou Visual C++ 6.0 COM pour la logique métier.

Les pages ASP sont totalement prises en charge dans Visual Basic 2005, et vous pouvez continuer à développer votre application en vous servant des pages ASP, des pages ASP.NET et des Web Forms. Les objets métier Visual Basic 6.0 et Visual C++ 6.0 peuvent être utilisés sans modification ou mis à niveau vers Visual Basic 2005.

Les classes Web n'existent plus dans Visual Basic 2005. Les applications Webclass existantes peuvent interagir avec les Web Forms de Visual Basic 2005 et les applications ASP ; elles peuvent également être mises à niveau vers des applications Web Forms.

Voir aussi

Concepts

Points à considérer avant d'effectuer la mise à niveau
Pourquoi une mise à niveau ?

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic
Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau relatives au langage

Comme il existe des différences de langage entre Visual Basic 6.0 et Visual Basic 2005, vous serez susceptible de modifier du code lors d'une mise à niveau d'une application.

Suivez les recommandations suivantes lorsque vous préparez une application Visual Basic 6.0 en vue de sa mise à niveau ou lorsque vous développez des applications Visual Basic 6.0 destinées à être mises à niveau ultérieurement.

Dans cette section

Recommandations de mise à niveau : utiliser la liaison anticipée et les conversions explicites

Fournit quelques conseils pour la conversion de la déclaration de variable et des types de données dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : utiliser le type de données Date pour l'enregistrement des dates

Fournit quelques conseils pour la gestion de la date dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : résoudre les propriétés par défaut sans paramètres

Fournit quelques conseils pour le référencement des propriétés par défaut dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : éviter la propagation de Null

Fournit quelques conseils pour la gestion des valeurs Null dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : utiliser des tableaux bornés à zéro

Fournit quelques conseils pour la création de tableaux dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : utiliser des constantes à la place des valeurs sous-jacentes

Fournit quelques conseils pour le référencement de constantes dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : éviter les tableaux et les chaînes de longueur fixe dans les types définis par l'utilisateur

Fournit quelques conseils pour les types définis par l'utilisateur dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : éviter les mots clés hérités (legacy)

Fournit quelques conseils pour l'utilisation de mots clés obsolètes dans les applications Visual Basic 6.0.

Recommandations de mise à niveau : ajuster les types de données pour les API Win32

Fournit quelques conseils les appels aux API Windows dans les applications Visual Basic 6.0.

Rubriques connexes

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Décrit comment s'assurer qu'une application est prête pour la mise à niveau.

Points à considérer avant d'effectuer la mise à niveau

Présente les restrictions à la mise à niveau et les points à prendre en considération avant de l'effectuer.

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Traite de la mise à niveau de vos applications Visual Basic 6.0.

Recommandations de mise à niveau : utiliser la liaison anticipée et les conversions explicites

Visual Basic 6.0 et Visual Basic 2005 prennent tous deux en charge les objets à liaison tardive (pratique consistant à déclarer une variable en tant que type de données **Object** et à l'assigner à une instance d'une classe au moment de l'exécution). Toutefois, durant le processus de mise à niveau, les objets à liaison tardive peuvent créer des problèmes lors de la résolution des propriétés par défaut ou dans les cas où le modèle objet sous-jacent a été modifié et les propriétés, méthodes et événements doivent être convertis. Prenons l'exemple d'un formulaire nommé `Form1` doté d'une étiquette, `Label1` ; le code Visual Basic 6.0 suivant affecterait à l'étiquette la légende "SomeText" :

```
Dim o As Object
Set o = Me.Label1
o.Caption = "SomeText"
```

Dans Visual Basic 2005, la propriété **Caption** d'un contrôle label est remplacée par la propriété Text. Lorsque votre code est mis à niveau, toutes les instances de la propriété **Caption** sont changées en **Text** ; toutefois, parce qu'un objet à liaison tardive est dénué de type, Visual Basic ne peut pas déterminer le type de cet objet ni détecter si certaines propriétés doivent être converties. Dans ces cas-là, vous devez changer vous-même le code après avoir opéré la mise à niveau.

Si vous réécrivez le code en utilisant des objets à liaison anticipée, il sera automatiquement mis à niveau :

```
Dim o As Label
Set o = Me.Label1
o.Text = "SomeText"
```

Dans la mesure du possible, déclarez des variables du type d'objet approprié au lieu de simplement les déclarer comme type de données **Object**.

Si vous faites appel aux variables **Object** et **Variant** dans votre code Visual Basic 6.0, nous vous recommandons d'utiliser des conversions explicites lorsque vous assignez les variables, effectuez des opérations sur les variables ou passez les variables à une fonction. Par exemple, dans le code suivant, la raison de l'utilisation de l'opération '+' n'est pas claire :

```
Dim Var1 As Variant
Dim Var2 As Variant
Dim Var3 As Variant
Var1 = "3"
Var2 = 4
'BAD: Should Var1 and Var2 be added as strings or integers?
Var3 = Var1 + Var2
```

L'exemple ci-dessus peut provoquer une erreur d'exécution dans Visual Basic 2005. Pour que le code fonctionne correctement, réécrivez la dernière ligne en utilisant les conversions explicites :

```
'GOOD: explicit conversion
Var3 = CInt(Var1) + CInt(Var2)
```

Visual Basic 2005 prend en charge les fonctions de surcharge en se basant sur le type du paramètre. Par exemple, la fonction **Environ** a désormais deux formes :

```
Environ( Expression As Integer) As String
Environ( Expression As String ) As String
```

Pour déterminer laquelle des deux fonctions il doit appeler, Visual Basic 2005 se base sur le type du paramètre. Si vous passez un entier à **Environ()**, la version avec entier est appelée ; en revanche, si vous passez une chaîne, c'est la version avec chaîne qui est appelée. Le code qui passe un type de données **Variant** ou **Object** à une fonction surchargée peut provoquer une erreur de compilation ou une erreur d'exécution. Si vous utilisez une conversion explicite, comme dans l'exemple suivant, votre code fonctionnera comme prévu une fois mis à niveau vers Visual Basic 2005 :

```
Dim a As String
Dim v As Variant
v = "Path"
'GOOD: explicit conversion
a = Environ(CStr(v))
```

L'utilisation de conversions explicites d'objets à liaison tardive constitue une bonne pratique de programmation. Elle permet de déterminer plus facilement l'objectif recherché par le code et simplifie le processus de mise à niveau de votre projet vers Visual Basic 2005.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : utiliser le type de données Date pour l'enregistrement des dates

Les versions précédentes de Visual Basic utilisaient le type de données **Double** pour le stockage et la manipulation des dates. Cette pratique n'est toutefois pas recommandée dans Visual Basic 2005, parce que les dates ne sont pas stockées en interne sous forme de valeurs de type Double. Par exemple, le code suivant est valide dans Visual Basic 6.0, mais peut provoquer une erreur de compilation dans Visual Basic 2005 :

```
Dim dbl As Double
Dim dat As Date
dat = Now
'BAD: Date can't be assigned to a double
dbl = dat
'BAD: Double can't be used in date functions
dbl = DateAdd("d", 1, dbl)
'BAD: CDate can't convert a double to a date
dat = CDate(dbl)
```

Le .NET Framework fournit les fonctions ToOADate et FromOADate pour une conversion de valeurs de type Double en type Date. Toutefois, lorsque votre projet est mis à niveau vers Visual Basic 2005, il est difficile de déterminer l'objectif recherché par le code utilisant des valeurs doubles pour le stockage des dates. Pour éviter que des modifications superflues soient apportées à votre code dans Visual Basic 2005, utilisez toujours le type de données **Date** pour le stockage des dates.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : résoudre les propriétés par défaut sans paramètres

Dans Visual Basic 6.0, de nombreux objets exposent des propriétés par défaut, lesquelles peuvent être omises comme raccourci de programmation. Par exemple, **TextBox** possède la propriété par défaut **Text** ; donc, au lieu d'écrire :

```
MsgBox Form1.Text1.Text
```

vous utilisez le raccourci :

```
MsgBox Form1.Text1
```

La propriété par défaut est résolue au moment de la compilation du code. En outre, vous pouvez utiliser les propriétés par défaut avec les objets à liaison tardive, comme dans l'exemple suivant :

```
Dim obj As Object
Set obj = Form1.Text1
MsgBox obj
```

Dans l'exemple de liaison tardive, la propriété par défaut est résolue au moment de l'exécution, et **MsgBox** affiche pour la propriété par défaut de **TextBox** la valeur **Text1**.

Visual Basic 2005 ne prend pas en charge les propriétés par défaut sans paramètre et, par conséquent, n'autorise pas ce raccourci de programmation. Lorsque votre projet est mis à niveau, Visual Basic 2005 résout les propriétés par défaut sans paramètre ; toutefois, les cas d'emploi de liaison tardive s'appuyant sur une résolution au moment de l'exécution ne peuvent pas être résolus automatiquement. Dans ces cas-là, vous devez changer vous-même le code après avoir opéré la mise à niveau. Le fait que beaucoup de bibliothèques implémentent des propriétés par défaut à l'aide d'une propriété nommée **_Default** constitue une complication supplémentaire. **_Default** agit comme un proxy, en passant des appels à la vraie propriété par défaut. Ainsi, lorsque votre projet sera mis à niveau, certaines propriétés par défaut seront résolues en **_Default**. Le code continuera à fonctionner comme avant, mais il sera moins compréhensible que du code écrit explicitement avec la propriété réelle. Voilà pourquoi vous devez éviter d'utiliser des propriétés par défaut sans paramètres dans votre code Visual Basic 6.0. Au lieu d'écrire :

```
Dim obj As Object
Set obj = Me.Text1
'BAD: Relying on default property
MsgBox obj
'BAD: Relying on default property
MsgBox Me.Text1
```

utilisez :

```
Dim obj As TextBox
Set obj = Me.Text1
'GOOD: Default property is resolved
MsgBox obj.Text
'GOOD: Default property is resolved
MsgBox Me.Text1.Text
```

Si Visual Basic 2005 ne prend pas en charge les propriétés par défaut sans paramètre, il prend en charge les propriétés par défaut avec paramètre. Pour comprendre ce qui différencie ces deux types de propriétés par défaut, retenez que les propriétés par défaut avec paramètres ont toujours un index. La propriété par défaut de l'objet ADO **recordset**, collection **Fields**, en est un exemple. Le code :

```
Dim rs As ADODB.Recordset
rs("CompanyName") = "SomeCompany"
rs!CompanyName = "SomeCompany"
```

est en réalité un raccourci de :

```
Dim rs As ADODB.Recordset
rs.Fields("CompanyName").Value = "SomeCompany"
rs.Fields!CompanyName.Value = "SomeCompany"
```

Dans ce cas, la propriété **Fields** est paramétrée et son utilisation dans Visual Basic 2005 est donc admise ; toutefois, la propriété par défaut de la propriété **Fields**, **Value**, est sans paramètre ; son usage correct dans Visual Basic 2005 est donc :

```
Dim rs As ADODB.Recordset
rs("CompanyName").Value = "SomeCompany"
rs!CompanyName.Value = "SomeCompany"
```

Cet exemple ainsi que la plupart des autres propriétés par défaut sont résolus pour vous lors de la mise à niveau du projet ; les résoudre dans Visual Basic 6.0 est donc simplement une bonne pratique de programmation. Toutefois, vous devez éviter d'utiliser les propriétés par défaut avec les types de données **Object** et **Variant** ; ces types ne pouvant pas être résolus, vous devez corriger vous-même le code dans le projet mis à niveau.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : éviter la propagation de Null

Les versions antérieures de Visual Basic prennent en charge la propagation de Null. La propagation de Null implique que lorsque **Null** est utilisé dans une expression, le résultat de l'expression est lui-même Null. Dans tous les cas de l'exemple suivant, le résultat de `V` est Null.

```
Dim V
V = 1 + Null
V = Null + Right$("SomeText", 1)
V = Right("SomeText", 0)
```

La propagation de Null n'est pas prise en charge dans Visual Basic 2005. L'instruction `1+Null` générera une incompatibilité de type dans Visual Basic 2005. En outre, alors que Visual Basic 6.0 possédait deux versions de la fonction **Left** — **Left$** retournant une chaîne, **Left** retournant un variant qui pouvait être Null — Visual Basic 2005 ne possède plus qu'une seule version, **Left**, qui retourne une chaîne dans tous les cas.

Par souci de compatibilité avec Visual Basic 6.0 et Visual Basic 2005, vous devez toujours écrire du code pour tester les Null au lieu de vous fier à la propagation de Null. En outre, dans Visual Basic 2005, les fonctions suivantes ne retournent plus Null :

Chr, **Command**, **CurDir**, **Date**, **Environ**, **Error**, **Hex**, **LCase**, **LTrim**, **Oct**, **Right**, **RTrim**, **Space**, **Str**, **Time**, **Trim**, **UCase**

La propagation de Null est couramment utilisée dans les applications de base de données qui impliquent de rechercher si un champ contient Null. Dans ce cas, vous pouvez vérifier les résultats à l'aide de la fonction **IsNull()** et effectuer l'action appropriée.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : utiliser des tableaux bornés à zéro

Dans Visual Basic 6.0, vous pouvez définir des tableaux avec des limites supérieure et inférieure correspondant à n'importe quel nombre entier. Vous pouvez également utiliser **ReDim** pour réassigner un variant sous forme de tableau. Pour une question d'interopérabilité avec les autres langages, les tableaux doivent avoir dans Visual Basic 2005 une limite inférieure égale à zéro et **ReDim** ne peut plus être utilisé à moins que la variable du tableau ait précédemment été déclarée à l'aide du mot clé **Dim**. Cette contrainte, si elle restreint la définition des tableaux, vous permet aussi de passer des tableaux entre Visual Basic 2005 et tout autre langage du .NET Framework. L'exemple suivant illustre cette contrainte :

```
'BAD: LBound must be 0
Dim a(1 To 10) As Integer
'BAD: Can't use ReDim without Dim
ReDim v(10)
'OK: Creates an array of 11 integers
Dim b(10) As Integer
'OK: Can ReDim previously Dimed variable
ReDim b(5) As Integer
```

Conséquence de ce changement, l'instruction **Option Base** est supprimée du langage.

Lorsque votre projet est mis à niveau vers Visual Basic 2005, toutes les instructions Option Base sont supprimées de votre code. Si le tableau est limité à zéro, il reste inchangé. Toutefois, si un tableau n'est pas limité à zéro, sa limite inférieure est supprimée et un avertissement est inséré dans le code, comme le montre l'exemple suivant :

```
Dim a(1 To 10) As Integer
```

Est changé en :

```
' UPGRADE_WARNING: Lower bound of an array was changed from 1 to 0.
Dim a(10) As Integer
```

Dans de nombreux cas, le code mis à niveau fonctionnera comme avant. Toutefois, si la logique de votre application se base sur une limite inférieure de 1, vous devez procéder à quelques modifications. Les instructions **Dim**, **ReDim** et **LBound** sont accompagnées d'avertissements afin de vous permettre de vérifier les modifications.

C'est pourquoi nous vous conseillons d'employer des tableaux bornés à zéro dans votre code Visual Basic 6.0, d'éviter de recourir à **ReDim** comme déclaration de tableau et d'éviter d'utiliser **Option Base** 1.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : utiliser des constantes à la place des valeurs sous-jacentes

Lorsque vous écrivez du code, essayez d'utiliser des constantes plutôt que de vous appuyer sur leurs valeurs sous-jacentes. Par exemple, pour agrandir un formulaire au moment de l'exécution, utilisez :

```
' GOOD: Constant name
Me.WindowState = vbMaximized
```

au lieu de :

```
'BAD: Underlying value
Me.WindowStyle = 2
'BAD: Variable
Me.WindowStyle = X
```

De la même façon, utilisez les valeurs **True** et **False** à la place de -1 et 0.

Dans Visual Basic 2005, les valeurs et parfois les noms de certaines propriétés et constantes ont changé. Lorsque votre projet est mis à niveau vers Visual Basic 2005, la plupart des constantes sont changées automatiquement ; toutefois, si vous utilisez des valeurs ou des variables sous-jacentes au lieu de noms de constantes, elles ne pourront souvent ne pas être mises à niveau automatiquement. L'utilisation des noms de constantes réduit le nombre de modifications que vous devez effectuer.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : éviter les tableaux et les chaînes de longueur fixe dans les types définis par l'utilisateur

Du fait des modifications introduites afin de rendre les tableaux et structures Visual Basic 2005 (précédemment désignés par le terme de "types définis par l'utilisateur") pleinement compatibles avec d'autres langages Visual Studio, les chaînes de longueur fixe ne sont plus prises en charge. En général, cela ne pose aucun problème, car il existe une classe de compatibilité présentant un comportement de chaîne de longueur fixe ; si bien que le code :

```
Dim FixedLengthString As String * 100
```

devient, après mise à niveau :

```
Dim FixedLengthString As New VB6.FixedLengthString(100)
```

Toutefois, les chaînes de longueur fixe provoquent un problème lorsqu'elles sont utilisées dans des structures. Ce problème est dû au fait que la classe de chaîne de longueur fixe n'est pas automatiquement créée en même temps que la structure. Pareillement, les tableaux de taille fixe ne sont pas créés en même temps que la structure.

Lorsque votre code est mis à niveau, les types définis par l'utilisateur avec chaîne ou tableau de longueur fixe sont convertis en structures et marqués d'un commentaire vous demandant d'initialiser la chaîne ou le tableau avant de référencer la structure dans le code. Toutefois, vous pouvez éviter cette modification en changeant vos types définis par l'utilisateur Visual Basic 6.0 de telle sorte qu'ils utilisent des chaînes à la place des chaînes de longueur fixe et des tableaux non initialisés à la place des tableaux de taille fixe. Par exemple :

```
Private Type uType
    anArray(5) As Integer
    aFixedString As String * 100
End Type
Sub SomeFunction()
    Dim aVariable As uType
End Sub
```

peut être changé en :

```
Private Type uType
    anArray() As Integer
    aFixedString As String
End Type
Sub SomeFunction()
    Dim aVariable As uType
    ReDim aVariable.anArray(5) As Integer
    aVariable.aFixedString = String$(100, " ")
End Sub
```

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : éviter les mots clés hérités (legacy)

Comme ils ont été supprimés du langage, vous devez éviter d'utiliser les mots clé suivants dans les projets Visual Basic 6.0 qui peuvent être mis à niveau vers Visual Basic 2005.

Mot clé	Explication
Def<*type***>**	Dans les versions antérieures de Visual Basic, **DefBool**, **DefByte**, **DefInt**, **DefLng**, **DefCur**, **DefSng**, **DefDbl**, **DefDec**, **DefDate**, **DefStr**, **DefObj** et **DefVar** sont utilisés dans la section des déclarations d'un module pour définir le type d'une plage de variables. Par exemple, `DefInt A-C` définit comme un entier toutes les variables commençant par la lettre A, B ou C. Au lieu d'utiliser les instructions **Def**<*type*>, vous devez déclarer explicitement les variables.
GoTo/GoSub calculé	Les instructions **GoTo**/**GoSub** calculées se présentent sous la forme suivante : `On x GoTo 100, 200, 300` Elles ne sont pas prises en charge dans Visual Basic 2005. Utilisez à leur place des instructions **If** ou des constructions **Select Case**.
GoSub/Return	Les instructions **GoSub** et **Return** ne sont pas prises en charge dans Visual Basic 2005. Dans la majorité des cas, vous pouvez les remplacer par des fonctions et des procédures.
Option Base 0|1	L'instruction **Option Base** était utilisée pour spécifier la limite inférieure par défaut d'un tableau, à savoir 0 ou 1. Cette instruction a été supprimée du langage, car Visual Basic 2005 ne prend en charge que les tableaux présentant une limite inférieure de zéro en mode natif. Les tableaux qui n'ont pas une limite inférieure de zéro sont pris en charge par la bibliothèque de compatibilité.
VarPtr, **ObjPtr**, **StrPtr**	**VarPtr**, **VarPtrArray**, **VarPtrStringArray**, **ObjPtr** et **StrPtr** sont des fonctions non documentées permettant d'obtenir l'adresse mémoire sous-jacente de variables. Ces fonctions ne sont pas prises en charge dans Visual Basic 2005.
LSet	Dans Visual Basic 6.0, l'instruction **LSet** peut être utilisée pour assigner une variable d'un certain type défini par l'utilisateur à une autre variable d'un autre type défini par l'utilisateur. Cette fonctionnalité n'est pas prise en charge dans Visual Basic 2005.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Recommandations de mise à niveau : ajuster les types de données pour les API Win32

Beaucoup d'API peuvent être utilisées exactement comme vous le faisiez dans Visual Basic 6.0, à cela près que vous devez ajuster vos types de données en conséquence. Le type de données **Long** Visual Basic 6.0 est remplacé dans Visual Basic 2005 par le type de données **Integer** et le type de données **Integer** Visual Basic 6.0 est remplacé dans Visual Basic 2005 par le type de données **Short**. Pendant la mise à niveau, ces modifications sont effectuées pour vous, et les API simples fonctionnent exactement comme dans Visual Basic 6.0. Par exemple :

```
Private Declare Function GetVersion Lib "kernel32" () As Long
Function GetVer()
    Dim Ver As Long
    Ver = GetVersion()
    MsgBox ("System Version is " & Ver)
End Function
```

Est changé en :

```
Private Declare Function GetVersion Lib "kernel32" () As Integer
Function GetVer()
    Dim Ver As Integer
    Ver = GetVersion()
    MsgBox("System Version is " & Ver)
End Function
```

En plus des mises à niveau des types de données numériques, Visual Basic 6.0 dispose d'un type de données de chaîne de longueur fixe qui n'est pas pris en charge dans Visual Basic 2005 et qui, après mise à niveau, devient une classe wrapper de chaîne de longueur fixe. Dans la plupart des cas, vous pouvez effectuer la même opération dans Visual Basic 6.0 en utilisant une chaîne normale. Par exemple :

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As Long
Function GetUser()
    Dim Ret As Long
    Dim UserName As String
    Dim Buffer As String * 25
    Ret = GetUserName(Buffer, 25)
    UserName = Left$(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox (UserName)
End Function
```

Peut être mieux écrit en utilisant une chaîne normale dont la longueur aura été fixée explicitement à 25 à la place d'une chaîne de longueur fixe :

```
Dim Buffer As String
Buffer = String$(25, " ")
```

Est mis à niveau dans Visual Basic 2005 comme suit :

```
Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Function GetUser()
    Dim Ret As Integer
    Dim UserName As String
    Dim Buffer As String
    Buffer = New String(CChar(" "), 25)
    Ret = GetUserName(Buffer, 25)
    UserName = Left(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox(UserName)
```

```
End Function
```

Dans certains cas, Visual Basic 2005 gère mieux le passage des chaînes aux API ; en effet, vous pouvez éventuellement déclarer la façon dont vous voulez que les chaînes soient passées à l'aide des mots clés **ANSI** et **UNICODE**.

Il existe trois cas dans lesquels vous devez impérativement effectuer certaines modifications. Le premier est le passage aux API de types définis par l'utilisateur contenant des chaînes de longueur fixe ou des tableaux d'octets. Dans Visual Basic 2005, vous devrez peut-être modifier votre code en ajoutant l'attribut **MarshallAs** (de l'espace de noms System.Runtime.InteropServices) à chaque chaîne de longueur fixe ou à chaque tableau d'octets du type défini par l'utilisateur. Le deuxième est l'utilisation du type de variable **As Any** dans une instruction **Declare**. Ce qui n'est pas pris en charge dans Visual Basic 2005. Les variables de type **As Any** servaient souvent à passer une variable qui était soit une chaîne soit **Null** ; vous pouvez remplacer cet usage propre à Visual Basic 6.0 en déclarant deux formes pour l'API, l'une avec des types Long, l'autre avec des types String. Par exemple, l'API **GetPrivateProfileString** a un paramètre *lpKeyName* de type **As Any**:

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias _
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal _
lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString _
As String, ByVal nSize As Long, ByVal lpFileName As String) As Long
```

Vous pouvez supprimer "**As Any**" en remplaçant **Declare** par deux versions, l'une qui accepte un type Long, l'autre un type String :

```
Private Declare Function GetPrivateProfileStringKey Lib "kernel32" Alias _
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal _
lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString _
As String, ByVal nSize As Integer, ByVal lpFileName As String) As Integer

Private Declare Function GetPrivateProfileStringNullKey Lib "kernel32" _
Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String, _
ByVal lpKeyName As Integer, ByVal lpDefault As String, ByVal _
lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName _
As String) As Integer
```

Lorsque vous souhaitez passer la valeur Null à l'API, vous utilisez la version **GetPrivateProfileStringNullKey**. De cette façon, la fonction peut être mise à niveau vers Visual Basic 2005.

Enfin, vous devrez peut-être effectuer des modifications si vous utilisez des API qui créent des threads, dérivent des classes Windows, raccordent des files d'attente de messages, etc. Certaines de ces fonctions peuvent provoquer une erreur d'exécution dans Visual Basic 2005. Parmi ces API, nombreuses sont celles qui ont des équivalents dans Visual Basic 2005 ou le .NET Framework. Vous devrez les corriger une par une.

Voir aussi

Autres ressources

Recommandations de mise à niveau relatives au langage

Bibliothèque de compatibilité Visual Basic 6.0

Si vous avez utilisé les versions précédentes de Visual Basic, vous noterez que Visual Basic 2005 présente de nombreuses modifications apportées au langage Visual Basic et que d'anciennes fonctions qui vous étaient familières ont apparemment disparu. Ces modifications étaient nécessaires pour rendre Visual Basic conforme au .NET Framework et à la spécification CLS (Common Language Specification).

Lorsqu'une application Visual Basic 6.0 est mise à niveau vers Visual Basic 2005, la conversion de certains éléments de code n'est pas possible, pour des raisons de différences syntaxiques ou architecturales. C'est pourquoi certaines fonctions de la bibliothèque de compatibilité Visual Basic 6.0 (**Microsoft.VisualBasic.Compatibility**) permettent au code de s'exécuter dans Visual Basic 2005 sans qu'il soit nécessaire d'effectuer des modifications trop importantes. Ces fonctions reproduisent le comportement de Visual Basic 6.0 tout en restant conformes à la spécification CLS. Chaque fois que cela est possible, l'outil de mise à niveau essaie de convertir directement votre code en fonctions dans l'espace de noms **Microsoft.VisualBasic** ou **System**.

Bien que les fonctions et objets dans l'espace de noms **Compatibility** aient été conçus pour prendre en charge l'outil de mise à niveau, rien ne vous empêche de les utiliser lors de la création d'une nouvelle application dans Visual Basic 2005. Toutefois, dans la plupart des cas, le .NET Framework fournit des fonctionnalités variées.

Voir aussi

Tâches

Comment : ajouter des contrôles de système de fichiers Visual Basic 6.0 à votre application

Référence

Référence d'espace de noms VisualBasic.Compatibility

Concepts

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Mise à niveau de Visual Basic 6.0

Assistant Mise à niveau de Visual Basic

Visual Basic 2005 marque une étape fondamentale entre le développement Windows traditionnel et l'élaboration d'applications multicouches et Web de nouvelle génération. C'est pourquoi vous devrez mettre à niveau votre code afin de tirer pleinement parti de Visual Basic 2005.

Processus de mise à niveau

Cette mise à niveau s'effectue automatiquement lorsque vous ouvrez un projet Visual Basic 6.0 dans Visual Basic 2005 : l'Assistant Mise à niveau vous guide pas à pas dans le processus et crée un nouveau projet Visual Basic 2005 (votre ancien projet est laissé intact). Ce processus ne fonctionne toutefois que dans un sens ; le nouveau projet Visual Basic 2005 ne peut pas être ouvert dans Visual Basic 6.0.

Une fois votre projet mis à niveau, le langage est modifié pour tenir compte des changements de syntaxe et vos formulaires Visual Basic 6.0 sont convertis en Windows Forms. Dans la majorité des cas, vous devrez apporter certaines modifications à votre code après sa mise à niveau. En effet, certains objets et fonctionnalités du langage n'ont pas d'équivalent dans Visual Basic 2005 ou ont un équivalent trop différent pour se prêter à une mise à niveau automatique. Une fois la mise à niveau effectuée, vous pouvez également opérer des modifications pour bénéficier de certaines nouvelles fonctionnalités de Visual Basic 2005.

Pour vous aider à apporter les modifications, une fois votre projet mis à niveau, Visual Basic 2005 y ajoute un rapport de mise à niveau qui comptabilise tous les problèmes. Il insère également des commentaires dans votre code mis à niveau pour vous alerter sur les instructions qui devront être changées. Ces commentaires étant affichés sous forme de tâches dans la nouvelle fenêtre **Liste des tâches**, vous pouvez facilement voir les modifications requises et naviguer jusqu'à l'instruction de code correspondante en double-cliquant simplement sur une tâche. Chaque tâche et élément du rapport de mise à niveau est associé à une rubrique de l'aide qui vous fournit un complément d'informations sur les modifications à effectuer et sur la meilleure façon de procéder.

Voir aussi

Concepts

Fin du processus de mise à niveau
Pourquoi une mise à niveau ?
Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Comment : mettre à niveau un projet à l'aide de l'Assistant Mise à niveau de Visual Basic

L'Assistant Mise à niveau de Visual Basic permet de mettre à niveau les projets Visual Basic 6.0 en vue de la poursuite éventuelle de leur développement dans Visual Basic 2005. Il ne modifie pas le projet Visual Basic 6.0 d'origine, mais crée un nouveau projet Visual Basic 2005 basé sur ce dernier.

Remarque

Bien que l'Assistant facilite le processus, il vous faudra parfois modifier certaines parties du code dans l'application mise à ni veau. Prenez le temps de vous familiariser avec Visual Visual Basic 2005 avant de démarrer le processus.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour mettre à niveau un projet Visual Basic 6.0 vers Visual Basic 2005

1. Ouvrez le projet dans Visual Basic 6.0. Dans le menu **Fichier**, cliquez sur **Créer <NomProjet>**. Exécutez l'application et vérifiez qu'elle ne contient pas d'erreur.
2. Démarrez Visual Studio 2005. Dans le menu **Fichier**, pointez sur **Ouvrir**, puis cliquez sur **Projet/Solution**.
3. Dans la boîte de dialogue **Ouvrir un projet**, naviguez jusqu'à l'emplacement du projet Visual Basic 6.0 et sélectionnez le fichier de projet (.vbp).

   Vous ouvrez ainsi l'Assistant Mise à niveau Visual Basic.
4. Sur la deuxième page de l'Assistant, sélectionnez une option de mise à niveau. Le choix d'options proposé varie en fonction du type du projet mis à niveau.

   Conseil

   Pour une explication des options disponibles, appuyez sur F1 dans l'Assistant.
5. Sur la troisième page de l'Assistant, entrez l'emplacement de création du nouveau projet.

   Remarque

   Chaque projet Visual Basic 2005 doit se trouver dans son propre dossier. Si vous spécifiez un dossier qui contient déjà des fichiers, vous serez invité à choisir un autre dossier.
6. Sur la quatrième page de l'Assistant, cliquez sur **Suivant** pour démarrer le processus de mise à niveau. Une fois le processus terminé, le nouveau projet apparaît dans l'**Explorateur de solutions**.
7. Dans l'**Explorateur de solutions**, double-cliquez sur le nœud **_UpgradeReport.htm** pour ouvrir le rapport de mise à niveau.
8. Examinez ce rapport. Toutes les erreurs doivent être corrigées pour que vous puissiez démarrer l'exécution de l'application. Tous les avertissements doivent être étudiés afin de vérifier qu'ils sont sans effets sur le comportement de l'application.

   Conseil

   Pour une explication d'une erreur ou d'un avertissement, cliquez sur sa description pour afficher une rubrique d'aide.
9. Dans le menu **Débogage**, cliquez sur **Démarrer**. Exécutez l'application en vérifiant qu'elle est exempte d'erreur et que

son comportement est identique à ce qu'il était dans Visual Basic 6.0.

Remarque
L'Assistant Mise à niveau ne peut pas détecter toutes les incompatibilités ; il est donc possible que vous rencontriez d'autres erreurs lorsque vous exécuterez l'application. Pour obtenir la liste des problèmes connus, consultez Dépannage des applications mises à niveau à partir de Visual Basic 6.0.

Voir aussi

Tâches

Comment : afficher le rapport de mise à niveau

Concepts

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

WebClasses pour les utilisateurs de Visual Basic 6.0

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Mettre à niveau le code Visual Basic 6, boîte de dialogue

Utilisez la boîte de dialogue **Mettre à niveau le code Visual Basic 6** pour mettre à niveau le code Visual Basic 6.0 à Visual Basic 2005 en tapant le code dans la fenêtre **Code** ou en le collant à partir de l'éditeur de code Visual Basic 6.0. Lorsque vous cliquez sur le bouton **Mettre à niveau**, le code mis à niveau est inséré dans l'éditeur de code de Visual Basic 2005, à l'emplacement du curseur.

En outre, les références COM requises par le code Visual Basic 6.0 peuvent être ajoutées sous l'onglet **Références** ; elles sont ainsi ajoutées aux références de votre projet Visual Basic 2005.

Pour accéder à la boîte de dialogue **Mettre à niveau le code Visual Basic 6**, cliquez sur **Mettre à niveau le code Visual Basic 6** dans le menu **Outils**.

Remarque
La commande **Mettre à niveau le code Visual Basic 6** n'est disponible que dans l'éditeur de code.

Liste des éléments d'interface utilisateur

Code

Le code Visual Basic 6.0 peut être saisi dans la fenêtre Code ou collé à partir de l'éditeur de code. Le code est conservé entre les différentes utilisations afin que vous puissiez le modifier.

Références

Affiche les références COM requises par votre code Visual Basic 6.0.

Nom de la référence

Affiche le nom convivial du composant COM. Seules les références vérifiées seront ajoutées au projet Visual Basic 2005.

Version

Affiche le numéro de version (majeur et mineur) du composant COM.

Chemin d'accès

Affiche le chemin d'accès et le nom de fichier du composant COM.

Ajouter une référence

Affiche la boîte de dialogue **Ajouter une référence**, contenant une liste de composants COM disponibles.

Mettre à niveau

Met à niveau le code saisi dans la fenêtre Code et l'insère dans l'éditeur de code de Visual Basic 2005 ; les références activées dans la liste Références sont également ajoutées au projet Visual Basic 2005.

Conseils d'utilisation de la boîte de dialogue Mettre à niveau le code Visual Basic 6

La boîte de dialogue **Mettre à niveau le code Visual Basic 6** ne peut pas convertir correctement tout le code. Pour optimiser votre utilisation de cet outil :

- Consultez Points à considérer avant d'effectuer la mise à niveau.
- Simplifiez la mise à niveau de votre code en suivant les recommandations de Recommandations de mise à niveau relatives au langage. La façon dont le code Visual Basic 6.0 est écrit peut avoir des conséquences sur la façon dont il sera mis à niveau.
- Testez le code dans Visual Basic 6.0 avant de tenter la mise à niveau. Si le code ne s'exécute pas dans Visual Basic 6.0, il ne s'exécutera pas dans Visual Basic 2005.
- Les signatures d'événement dans Visual Basic 2005 sont différentes de leurs équivalents dans Visual Basic 6.0. Lorsque vous entrez du code pour les procédures événementielles de formulaire ou de contrôle, commencez par ajouter la déclaration de la procédure dans Visual Basic 2005, puis mettez à niveau uniquement le corps de la procédure.
- La boîte de dialogue **Mettre à niveau le code Visual Basic 6** est destinée à mettre à niveau de petites portions de code. Le code tapé dans la boîte de dialogue détermine le contexte de la mise à niveau.

Par exemple, l'instruction `L.Caption = "MyCaption"` ne sera pas traduite parce que l'outil ne peut pas déterminer le type de "`L`" pour résoudre la propriété `Caption`. L'insertion d'une déclaration pour `L` (par exemple `Dim L As Label`) avant l'instruction produit une mise à niveau réussie.

- Vérifiez que la syntaxe de votre code Visual Basic 6.0 est correcte, car une syntaxe incorrecte peut provoquer une erreur `Untranslated statement in WrapperSub`.
- Pour de meilleures performances, mettez à niveau le code dans les modules plutôt que dans les modules Form. Si vous devez mettre à niveau un formulaire, utilisez l'Assistant Mise à niveau.
- Si vous collez le code qui contient des caractères Unicode, une boîte de dialogue apparaîtra, vous demandant si vous souhaitez supprimer ceux-ci. Si vous sélectionnez OK, le code sera analysé et tous les caractères Unicode seront supprimés.

Voir aussi

Tâches

Comment : mettre à niveau du code Visual Basic 6.0 à l'aide de la boîte de dialogue Mettre à niveau le code Visual Basic 6

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Comment : mettre à niveau du code Visual Basic 6.0 à l'aide de la boîte de dialogue Mettre à niveau le code Visual Basic 6

La boîte de dialogue **Mettre à niveau le code Visual Basic 6** est un outil permettant de convertir le code Visual Basic 6.0 en code Visual Basic 2005. Contrairement à l'Assistant Mise à niveau de Visual Basic, qui met à niveau l'ensemble d'une application, la boîte de dialogue Mettre à niveau le code Visual Basic 6 est disponible au moment du design pour convertir de petites quantités de code lors du développement de votre application Visual Basic 2005.

La boîte de dialogue **Mettre à niveau le code Visual Basic 6** est utile pour deux raisons :

- Elle permet l'utilisation du code Visual Basic 6.0 existant, en le collant dans l'outil.
- Vous pouvez entrer du code à l'aide de la syntaxe Visual Basic 6.0 afin d'apprendre l'équivalent Visual Basic 2005.

Après avoir exécuté l'outil, le code Visual Basic 2005 est inséré dans l'éditeur de code à l'emplacement actuel du curseur. En cas de problème de conversion, des commentaires sont insérés et comportent des liens vers des pages d'aide qui expliquent le problème ainsi que les solutions possibles.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour mettre à niveau du code Visual Basic 6.0

1. Dans l'éditeur de code, placez le pointeur de la souris à l'emplacement où vous voulez insérer du code.
2. Dans le menu **Outils**, choisissez **Mettre à niveau le code Visual Basic 6**.

   La boîte de dialogue **Mettre à niveau le code Visual Basic 6** s'affiche.
3. Dans la boîte de dialogue **Mettre à niveau le code Visual Basic 6**, collez le code Visual Basic 6.0 existant ou entrez le code à l'aide de la syntaxe Visual Basic 6.0.

   Remarque

   Si vous collez le code qui contient des caractères Unicode, une boîte de dialogue apparaîtra, vous demandant si vous souhaitez supprimer ceux-ci. Si vous choisissez **OK**, le code sera analysé et tous les caractères Unicode seront supprimés.
4. Si votre code fait référence à un composant COM, sélectionnez l'onglet **Références** et cliquez sur **Ajouter une référence**, puis sélectionnez la référence dans la boîte de dialogue **Ajouter une référence**.
5. Cliquez sur **Mettre à niveau** pour convertir le code et l'insérer dans l'éditeur de code.

   Toute référence COM sera ajoutée au nœud **Références** de votre projet dans l'**Explorateur de solutions** et une instruction **Imports** de niveau du projet sera ajoutée à l'espace de noms **Microsoft.VisualBasic.Compatibility**.

Voir aussi

Référence

Mettre à niveau le code Visual Basic 6, boîte de dialogue

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Comment : émuler un contrôle Tri-state Visual Basic 6.0 dans une application mise à niveau

Dans Visual Basic 6.0, les **Picture**, **DownPicture** et les propriétés **DisabledPicture** sont utilisés pour afficher des images différentes selon l'état d'un contrôle **CheckBox**, **CommandButton** ou **OptionButton**. Par exemple, lorsqu'un contrôle **CheckBox** est activé, l'image **DownPicture** est affichée ; si le contrôle est désactivé, l'image **DisabledPicture** est affichée.

Dans Visual Basic 2005, vous pouvez obtenir le même effet à l'aide d'un contrôle ImageList comme décrit dans l'exemple suivant.

Remarque

Vérifiez d'abord votre application Visual Basic 6.0. Si les propriétés **DownPicture** et **DisabledPicture** ne sont pas définies a u moment du design ou de l'exécution, le comportement devrait rester le même dans Visual Basic 2005.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Ajout d'un contrôle ImageList

Suivez les étapes suivantes pour modifier l'application mise à niveau si les propriétés **DownPicture** ou **DisabledPicture** sont définies.

Pour émuler un contrôle Tri-state

1. Déterminez les noms de fichiers et les emplacements des images assignées aux propriétés **Picture**, **DownPicture** et **DisabledPicture**, si nécessaire, copiez-les vers votre ordinateur de développement.
2. À partir de la **boîte à outils**, ajoutez un contrôle **ImageList** à votre formulaire.
3. Dans la fenêtre **Propriétés**, sélectionnez la propriété Images.
4. Dans l'**Éditeur de la collection Image**, ajoutez trois images à utiliser pour le **Picture**, puis le **DownPicture**, puis le **DisabledPicture**.
5. Si l'une de ces propriétés a été définie au moment de l'exécution, supprimez le code correspondant. Si les propriétés ont été définies au moment du design, ajoutez le code suivant à l'événement Load du formulaire :

   VB

   ```
   ' Assign the first image (Picture) to the Image property.
   CheckBox1.Image = ImageList1.Images(0)
   ```

6. Pour afficher l'image **DownPicture** au moment de l'exécution, ajoutez le code suivant à l'événement CheckedChanged du contrôle CheckBox.

   VB

   ```
   If CheckBox1.Checked = True Then
     ' Assign the second image (DownPicture) to the Image property.
     CheckBox1.Image = ImageList1.Images(1)
   Else
     ' Assign the first image (Picture) to the Image property.
     CheckBox1.Image = ImageList1.Images(0)
   End If
   ```

7. Pour afficher l'image **DisabledPicture** au moment de l'exécution, ajoutez le code suivant à l'événement EnabledChanged du contrôle **CheckBox**.

   VB

```
If CheckBox1.Enabled = False Then
  ' Assign the third image (DisabledPicture) to the Image property.
  CheckBox1.Image = ImageList1.Images(2)
ElseIf CheckBox1.Checked = True Then
  ' Assign the second image (DownPicture) to the Image property
  CheckBox1.Image = ImageList1.Images(1)
Else
  ' Assign the first image (Picture)to the Image property
  CheckBox1.Image = ImageList1.Images(0)
End If
```

   L'application devrait désormais avoir un comportement en tout point identique à celui qu'elle avait dans Visual Basic 6.0.

Voir aussi

Concepts

Propriété Style pour les utilisateurs de Visual Basic 6.0
Contrôle CheckBox pour les utilisateurs de Visual Basic 6.0
Contrôle CommandButton pour les utilisateurs de Visual Basic 6.0
Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0
Contrôle OptionButton pour les utilisateurs de Visual Basic 6.0
Contrôles utilisateur pour les utilisateurs de Visual Basic 6.0

Mise à niveau de projets WebClass vers Visual Basic 2005

Les projets WebClass Visual Basic 6.0 (également appelés projets d'application IIS) peuvent être mis à niveau vers des projets de site Web ASP.NET à l'aide de l'Assistant Mise à niveau. Le processus de mise à niveau des projets WebClass est pour l'essentiel identique à celui de tout autre type de projet. Cependant, il est important d'être conscient de certains points.

Mise à niveau des classes Web

Lors de la mise à niveau d'un projet WebClass, le projet est créé par défaut avec le nouveau nom *projectname*.NET (où *projectname* est le nom du projet Visual Basic 6.0).

Lorsqu'un projet WebClass Visual Basic 6.0 est mis à niveau vers ASP.NET, le fichier .asp du projet est mis à niveau vers un fichier .aspx. Les références au fichier .asp contenues dans un fichier modèle HTML ne sont pas converties automatiquement en références .aspx. Ces références ne sont pas mises à niveau car un fichier modèle peut contenir des références à d'autres fichiers .asp qui ne faisaient pas partie du projet WebClass.

De plus, lors de la copie des fichiers dans le nouveau répertoire de projet au cours de la mise à niveau, seuls les fichiers modèles HTML sont copiés. Aucun autre fichier .html ou fichier image n'est copié dans le nouveau répertoire.

Lors de l'ajout de fichiers HTML à un projet site Web ASP.NET, ceux-ci sont ajoutés par défaut sous forme de fichiers de contenu. Lorsqu'un projet WebClass est mis à niveau, les fichiers HTML sont ajoutés en tant que ressources incorporées. Si vous ajoutez des fichiers HTML au projet après la mise à niveau, vous devez définir leur propriété **Action de génération** sur **Ressource incorporée** afin de les rendre visibles pour l'application.

Conseil
Vous pouvez tout de même référencer un fichier HTML qui n'est pas incorporé en indiquant le chemin d'accès complet du fichier.

Si vous n'avez jamais créé d'application de site Web ASP.NET, avant de mettre à niveau votre premier projet WebClass Visual Basic 6.0, vous devrez définir l'objet **StartUp** du projet. Pour ce faire, cliquez avec le bouton droit sur le fichier .aspx et sélectionnez **Définir comme page de démarrage** dans le menu contextuel avant d'exécuter le projet mis à niveau.

Voir aussi

Concepts

WebClasses pour les utilisateurs de Visual Basic 6.0

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Fin du processus de mise à niveau

Après avoir exécuté l'Assistant Mise à niveau, votre projet mis à niveau est ouvert dans Visual Basic 2005. Le projet apparaît dans l'**Explorateur de solutions**; il inclut les versions mises à niveau de tous ses fichiers projet ainsi qu'un rapport de mise à niveau.

En général, pour garantir que le projet fonctionnera correctement dans Visual Basic 2005, il vous faudra procéder à quelques ajustements complémentaires. Ainsi, trois types d'éléments de travail doivent être complétés :

- Les erreurs UPGRADE_ISSUE sont des éléments qui empêchent la compilation de l'application. Vous devez corriger ces erreurs avant d'exécuter le projet. Elles sont signalées dans le rapport de mise à niveau et apparaissent aussi dans la liste des tâches.
- Les erreurs UPGRADE_TODO sont des éléments qui n'empêchent pas la compilation du projet, mais provoquent des erreurs au moment de l'exécution. Vous devez corriger ces erreurs avant d'exécuter le projet. Elles sont signalées dans le rapport de mise à niveau et apparaissent sous forme d'éléments dans la **Liste des tâches**, ainsi que de commentaires dans le code mis à niveau.
- Les erreurs UPGRADE_WARNING sont des éléments qui n'empêchent pas la compilation du projet, mais qui peuvent provoquer des erreurs au moment de l'exécution. Vous devez en rechercher l'origine et les corriger en conséquence. Elles sont signalées dans le rapport de mise à niveau et apparaissent sous forme d'éléments dans la **Liste des tâches**, ainsi que de commentaires dans le code mis à niveau.
- Les éléments UPGRADE_NOTE représentent des modifications du code ou des différences de comportement majeures qui peuvent avoir des conséquences indésirables. Vous devez les lire et décider si des modifications s'imposent. Elles sont signalées sous forme de commentaires dans le code.

Pour compléter le processus de mise à niveau, vous devez rechercher dans la **Liste des tâches** d'éventuels avertissements ou erreurs et dans votre code tous les commentaires qui commenceraient par "UPGRADE_NOTE".

Conseil
Chaque ligne de commentaire insérée dans votre code pendant la mise à niveau contient un lien hypertexte. Si vous cliquez sur ce lien (CTRL + clic), vous êtes dirigé vers une page d'aide qui vous explique le problème et vous indique les actions à entreprendre pour y remédier. Ce lien se trouve à la fin de la ligne de commentaire ; il vous faudra peut-être faire défiler la ligne pour le voir.

Voir aussi

Concepts

Assistant Mise à niveau de Visual Basic

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Rapport de mise à niveau Visual Basic

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, un rapport de mise à niveau est ajouté au projet dans l'**Explorateur de solutions**. Ce rapport contient des informations sur le processus de mise à niveau, ainsi qu'une liste des problèmes détectés et qui devront être corrigés pour que votre projet puisse être exécuté.

Détails du rapport

Le rapport de mise à niveau est au format HTML ; vous pouvez l'afficher dans l'environnement de développement Visual Basic 2005 en double-cliquant sur le fichier _UpgradeReport.htm dans l'**Explorateur de solutions** ou dans un navigateur externe en sélectionnant **Afficher dans le navigateur** dans le menu contextuel.

La première partie de ce rapport contient des informations générales sur la mise à niveau, et notamment les paramètres utilisés et l'emplacement des nouveaux fichiers projet.

Le reste du rapport contient une liste des problèmes généraux, ainsi qu'une liste de tous les fichiers mis à niveau. En développant la section de chaque fichier, vous affichez une liste détaillée des problèmes auxquels vous devez remédier. Chaque problème contient des informations indiquant sa description, son niveau de gravité ainsi que l'emplacement du code à corriger.

Voir aussi

Concepts

Fin du processus de mise à niveau

Assistant Mise à niveau de Visual Basic

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Comment : afficher le rapport de mise à niveau

Lorsqu'un projet Visual Basic 6.0 est mis à niveau vers Visual Basic 2005, un rapport de mise à niveau est créé et placé dans le dossier du nouveau projet. Ce rapport contient un résumé du processus de mise à niveau ainsi que des informations sur les éventuels problèmes et erreurs rencontrés.

Remarque
Le rapport de mise à niveau n'est généré que pour les projets mis à niveau à l'aide de l'Assistant Mise à niveau ; aucun rapport n'est généré pour les projets mis à niveau à partir de la ligne de commande.

Le rapport de mise à niveau peut être affiché dans l'environnement de développement intégré (IDE) Visual Basic 2005 ou dans n'importe quel navigateur Web.

Remarque
Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour afficher le rapport de mise à niveau dans Visual Studio 2005

- Dans l'**Explorateur de solutions**, double-cliquez sur le nœud **_UpgradeReport.htm** pour ouvrir le rapport de mise à niveau.

Pour afficher le rapport de mise à niveau dans un navigateur Web

1. Dans l'**Explorateur de solutions**, sélectionnez le nœud **_UpgradeReport.htm**.
2. Dans le menu **Fichier**, choisissez **Naviguer avec**.
3. Dans la boîte de dialogue **Naviguer avec**, sélectionnez le navigateur Web.

Voir aussi

Concepts

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Comment : mettre à niveau un projet à partir de la ligne de commande

En plus de l'Assistant Mise à niveau de Visual Basic, il existe un outil de ligne de commande, Vbupgrade.exe, qui permet de mettre à niveau les projets Visual Basic 6.0 vers Visual Basic 2005.

Remarque

Vbupgrade.exe ne se trouve pas dans le chemin d'accès par défaut. Vous devez par conséquent l'ajouter à ce chemin d'accès ou vous placer dans son répertoire d'installation avant de l'exécuter à partir de la ligne de commande. L'emplacement d'installation par défaut est C:\Program Files\Microsoft Visual Studio 8\Vb\VBUpgrade.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour mettre à niveau un projet à partir de la ligne de commande

1. Ouvrez le projet dans Visual Basic 6.0. Dans le menu **Fichier**, cliquez sur **Créer <nomprojet>**. Exécutez l'application et vérifiez qu'elle ne contient pas d'erreur.
2. À l'invite de la ligne de commande, tapez **Vbupgrade <nomprojet>** (où <*projectname*> est le chemin d'accès complet de l'application Visual Basic 6.0) suivi des arguments de ligne de commande requis.

   Remarque

   Pour plus d'informations sur la syntaxe et les options de ligne de commande de Vbupgrade, consultez Syntaxe de ligne de commande de Vbupgrade.

3. Ouvrez le projet résultant dans Visual Basic 2005. Examinez le code (des remarques y sont insérées pendant la mise à jour pour vous signaler les problèmes potentiels). Toutes les erreurs doivent être corrigées pour que vous puissiez démarrer l'exécution de l'application. Tous les avertissements doivent être étudiés afin de vérifier qu'ils sont sans effets sur le comportement de l'application.

   Conseil

   Pour obtenir l'explication d'une erreur ou d'un avertissement, cliquez sur sa description pour afficher l'aide.

4. Dans le menu **Déboguer**, cliquez sur **Démarrer**. Exécutez l'application en vérifiant qu'elle est exempte d'erreur et que son comportement est identique à ce qu'il était dans Visual Basic 6.0.

   Remarque

   L'Assistant Mise à niveau ne peut pas détecter toutes les incompatibilités ; il est donc possible que vous rencontriez d'autres erreurs lorsque vous exécuterez l'application. Pour obtenir la liste des problèmes connus, consultez Dépannage des applications mises à niveau à partir de Visual Basic 6.0.

Voir aussi

Tâches

Comment : afficher le rapport de mise à niveau

Référence

Syntaxe de ligne de commande de Vbupgrade

Concepts

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Syntaxe de ligne de commande de Vbupgrade

L'outil de ligne de commande Vbupgrade.exe met à niveau une application Visual Basic 6.0 vers Visual Basic 2005.

```
vbupgrade[.exe] inputname [/out outputname] [/verbose [/nolog [/logfile logfilename]]]
vbupgrade[.exe] /?
```

Arguments

inputname

Requis.

Chemin d'accès et nom de fichier du projet Visual Basic 6.0 à mettre à niveau.

/out *outputname*

Spécifie le chemin d'accès du dossier dans lequel le projet Visual Basic 2005 a été créé. Le chemin par défaut est \OutDir.

/verbose

Affiche toutes les sorties vers la fenêtre d'invite de commandes pendant la mise à niveau.

/nolog

Inhibe la création d'un fichier journal pendant la mise à niveau.

/logfile *logfilename*

Chemin d'accès et nom du fichier journal créé pendant la mise à niveau. Si le chemin d'accès et le nom de fichier ne sont pas spécifiés, un fichier journal est créé dans le même dossier que celui du projet Visual Basic 2005. Le nom de fichier par défaut est *ProjectFileName*.log, où *ProjectFileName* est le nom du fichier du projet.

/?

Affiche une liste d'options de ligne de commande.

Notes

Les chemins d'accès ou les noms de fichiers contenant des espaces doivent être placés entre guillemets.

Voir aussi

Tâches

Comment : mettre à niveau un projet à partir de la ligne de commande

Référence

Assistant Mise à niveau

Dépannage des applications mises à niveau à partir de Visual Basic 6.0

Les outils de mise à niveau de Visual Basic 2005 sont soigneusement conçus pour détecter et répertorier tous les problèmes des applications mises à niveau. Toutefois, certains d'entre eux restent impossibles à détecter. Cette page répertorie certains problèmes connus non détectés par les outils de mise à niveau et explique comment y remédier.

Impossible d'accéder aux liens d'aide après une mise à niveau de Visual Studio

Lorsque vous travaillez avec une application mise à niveau à partir d'une version antérieure de Visual Studio, les liens d'aide insérés par l'outil de mise à niveau peuvent générer l'erreur "Page non trouvée". Ce problème est dû au fait que les liens font référence à la collection d'aide de la version antérieure et que le format des liens à l'aide a changé.

Pour résoudre ce problème, vous pouvez copier la chaîne d'erreur et utiliser la fonction de recherche de l'aide pour rechercher la rubrique. Notez que certaines chaînes d'erreur contiennent des variables et que vous devrez donc faire une recherche sur une chaîne partielle.

Différence de comportement des chaînes de longueur fixe dans les types définis par l'utilisateur

Dans Visual Basic 6.0, les chaînes assignées à une chaîne de longueur fixe dans un type défini par l'utilisateur sont automatiquement tronquées lorsqu'elles dépassent la longueur fixée. Une fois effectuée la mise à niveau vers Visual Basic 2005, les chaînes ne seront plus tronquées. Il se peut donc que vous obteniez des résultats incorrects.

Remarque

Pendant la mise à niveau, l'attribut **VBFixedString** est ajouté aux chaînes de longueur fixe dans les types définis par l'utilisateur. Cet attribut permet aux fonctions de fichier de la bibliothèque de compatibilité Visual Basic de les traiter comme des chaînes de longueur fixe.

Pour résoudre ce problème, recherchez un code assignant une chaîne à la chaîne de longueur fixe, ajoutez du code pour vérifier la longueur de la chaîne et tronquez-la si nécessaire :

VB

```
' Before
MyString = "1234567"
MyStruct.FixedString5 = MyString

' After
MyString = "1234567"
If Len(MyString) > 5 Then
  MyString = Microsoft.VisualBasic.Left(MyString, 5)
End If
MyStruct.FixedString5 = MyString
```

La fermeture d'un formulaire appelle Dispose

Dans Visual Basic 6.0, il est possible de décharger un formulaire puis de le recharger ultérieurement en appelant la méthode **Show**. Dans Visual Basic 2005, la méthode **Close** de fermeture d'un formulaire appelle la méthode **Dispose** de manière à ce que le formulaire soit soumis automatiquement à l'opération de garbage collection. Cette opération peut provoquer de légères différences de comportement, parfois difficiles à détecter.

- Dans Visual Basic 2005, si vous appelez la méthode **Show** pour un formulaire ayant déjà été déchargé, vous obtiendrez une nouvelle instance du formulaire. Tous les paramètres de propriété de la classe de base seront perdus.
- **Dispose** n'est pas appelé automatiquement pour les formulaires avec affichage modal. Dans certains cas, il est utile d'appeler **Dispose** afin de libérer des ressources.

Les appels à liaison tardive à des objets COM peuvent provoquer des erreurs d'incompatibilité de type

Dans Visual Basic 6.0, lorsqu'un objet à liaison tardive COM est passé comme paramètre à un appel à liaison tardive, l'objet est forcé à correspondre à une **Variant** de type **Nothing**. Une fois mis à niveau vers Visual Basic 2005, les objets COM déclarés comme étant de type **Object** sont traités de la même manière que des **Variants** (qui sont toujours convertis en type **Object**

lors de la mise à niveau) ; ces objets sont marshalés en type de variant **Empty**. Ceci déclenche une erreur d'incompatibilité de type dans Visual Basic 2005.

Pour résoudre ce problème, assurez-vous que tous les objets sont à liaison anticipée.

Les valeurs retournées par Err.Number peuvent être différentes

Dans certains cas, les erreurs retournées par Visual Basic 2005 peuvent être différentes de celles retournées par Visual Basic 6.0. Dans le cas du code de gestion des erreurs qui se basait sur les valeurs retournées par **Err.Number**, cette différence peut modifier le comportement de votre application.

Le code suivant en donne une illustration :

```
' Visual Basic 6.0
On Local Error GoTo Result
Dim x() As Boolean
Dim y As Variant

y = x(10)

Result:
If Err.Number = 9 Then
   ' Do something.
Else
   ' Do something else.
End If
```

Avant de procéder à la mise à niveau, **Err.Number** retourne toujours 9 (indice hors limites) et exécute la première partie de l'instruction **If**. Après la mise à niveau, Err.Number retourne 91 (variable objet ou bloc With non défini) et exécute la clause **Else**. En effet, dans Visual Basic 2005, tout tableau doit être initialisé afin de pouvoir être référencé, alors que dans Visual Basic 6.0, les tableaux sont initialisés lors de leur déclaration.

Si, dans votre code, vous faites référence aux valeurs retournées par **Err.Number**, testez attentivement les résultats et modifiez votre code en conséquence.

Voir aussi

Concepts

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Utilisation de Visual Basic 6.0 et de la version actuelle de Visual Basic

Visual Basic 2005 et Visual Basic 6.0, ainsi que les applications créées au moyen de l'un ou l'autre de ces langages, peuvent être installés et simultanément exécutées sur le même ordinateur.

Les composants développés en Visual Basic 2005 peuvent interagir avec des composants COM développés dans des versions antérieures de Visual Basic et d'autres langages. Par exemple, vous pouvez faire glisser un contrôle ActiveX développé en Visual Basic 6.0 sur un Windows Form Visual Basic 2005, utiliser un objet de COM Visual Basic 6.0 issu d'une bibliothèque de classes Visual Basic 2005 ou ajouter une référence à une bibliothèque Visual Basic 2005 à un fichier exécutable Visual Basic 6.0.

Les composants compilés à l'aide de Visual Basic 2005 présentent de légères différences à l'exécution par rapport à ceux qui ont été compilés à l'aide de Visual Basic 6.0. Comme les objets Visual Basic 2005 sont libérés par l'opération de garbage collection (le "ramasse-miettes"), il peut y avoir un décalage entre le moment où les objets sont explicitement détruits et celui où ils sont effectivement supprimés de la mémoire. Il existe des différences supplémentaires, telles que le type de données et d'autres modifications apportée au langage. Ces différences font qu'en fin de compte, les applications Visual Basic 2005 affichent un comportement à l'exécution similaire, mais non identique, à celui des applications Visual Basic 6.0.

De plus, Visual Basic 2005 rend superflue la compatibilité binaire entre les composants Visual Basic 2005 et les composants Visual Basic 6.0. Les composants ont désormais un système de déploiement et de versioning plus robuste, les fichiers peuvent être déployés par simple copie dans un répertoire et la mise à niveau vers une nouvelle version est aussi simple que de remplacer un ancien fichier par un nouveau. Votre seule tâche consiste à veiller à ce que les classes et les méthodes soient compatibles avec les versions antérieures.

Voir aussi

Concepts

Pourquoi une mise à niveau ?

Préparation d'une application Visual Basic 6.0 en vue de sa mise à niveau

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Procédure pas à pas : mise à niveau d'une application Visual Basic 6.0 vers la version actuelle de Visual Basic

Cette procédure pas à pas permet de mettre à niveau une application graphique exemple Visual Basic vers Visual Basic 2005. En plus d'illustrer le processus de mise à niveau, cette procédure pas à pas met en évidence les différences qui existent entre les architectures graphiques de Visual Basic 6.0 et de Visual Basic 2005. Cette procédure peut également servir d'introduction à la gestion des problèmes d'architecture rencontrés dans vos applications.

Dans le cas présent, l'application exemple est un simple formulaire contenant deux boutons, une zone d'image, une minuterie et un contrôle Image caché. Bien que cet exemple ne soit pas représentatif de la plupart des applications que vous souhaitez mettre à niveau, il présente de nombreux problèmes complexes que vous pourriez rencontrer lors de la mise à niveau de vos propres applications.

Remarque
Cette procédure pas à pas nécessite que Visual Basic 6.0 soit installé sur votre ordinateur de développement.

Pour créer l'application Visual Basic 6.0

1. Ouvrez Visual Basic 6.0. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet**, choisissez **EXE standard** et cliquez sur **OK**.
3. Ajoutez un contrôle **PictureBox** au formulaire, et dimensionnez-le de manière à couvrir la plus grande partie du formulaire, tout en ménageant de l'espace dans la partie inférieure.
4. Attribuez à la propriété **AutoRedraw** la valeur **True**, à la propriété **DrawStyle** la valeur **0-Solid**, et à la propriété **FillStyle** la valeur **0-Solid**.
5. Ajoutez deux contrôles **CommandButton** sous le contrôle **PictureBox**.
6. Sélectionnez le premier **CommandButton**, puis attribuez à la propriété **Name** la valeur `ClearPictureBox` et à la propriété **Caption** la valeur `Clear`.
7. Sélectionnez le deuxième **CommandButton**, puis attribuez à la propriété **Name** la valeur `ShowImage` et à la propriété **Caption** la valeur `Show Image`.
8. Ajoutez un contrôle **Image** au formulaire puis attribuez à la propriété **Name** la valeur `sourceImage` et à la propriété **Visible** la valeur **False**.
9. Définissez la propriété **Picture** du contrôle **Image** à n'importe quelle image bitmap. Vous pouvez utiliser une des images dans votre dossier Mes images.
10. Ajoutez un contrôle **Timer** au formulaire. Attribuez à la propriété **Enabled** la valeur **False** et à la propriété **Interval** la valeur 25.
11. Double-cliquez sur le formulaire pour ouvrir l'éditeur de code, et entrez le code suivant.

```
Option Explicit
Private LeftPos As Double

Private Sub ClearPictureBox_Click()
    Picture1.Cls
End Sub

Private Sub ShowImage_Click()
    LeftPos = 1
    Me.Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()
```

```
    If LeftPos <= 0 Then
        Me.Timer1.Enabled = False
        Picture1.Print "Visual Basic ROCKS!"
    Else
        LeftPos = LeftPos - 0.01
        Picture1.Cls
        Picture1.PaintPicture sourceImage, LeftPos * _
Picture1.Width, 0
    End If
End Sub
```

12. Appuyez sur F5 pour exécuter l'application. Cliquez sur les boutons pour observer le comportement et exécutez le code pour pas à pas pour examiner son fonctionnement.
13. Dans le menu **Fichier**, choisissez **Enregistrer le projet sous**.
14. Dans la boîte de dialogue **Enregistrer le fichier sous**, enregistrez le formulaire en tant que `PicForm.frm` et enregistrez le projet en tant que `Drawing.vbp`.

Pour exécuter l'Assistant Mise à niveau

1. Ouvrez Visual Basic 2005. Dans le menu **Fichier**, sélectionnez **Ouvrir un projet**.
2. Dans la boîte de dialogue **Ouvrir un projet**, localisez le fichier **Drawing.vbp**, puis ouvrez-le.

   Vous lancez ainsi l'Assistant Mise à niveau Visual Basic. La première page de l'Assistant vous en explique le rôle.
3. Cliquez sur **Next** pour passer à la seconde page de l'Assistant. Cette page présente les options de mise à niveau ; dans le cas présent, aucune option applicable n'est disponible.
4. Cliquez sur **Next** pour passer à la troisième page de l'Assistant. Cliquez sur **Next**pour accepter l'emplacement par défaut du nouveau projet, ou indiquez un emplacement différent. L'emplacement défini par défaut est un nouveau dossier situé juste en dessous du dossier du projet Visual Basic 6.0.

   Si un message vous invite à créer un dossier, choisissez **Oui**.
5. Sur la quatrième page de l'Assistant, cliquez sur **Next** pour démarrer la mise à niveau.

   Une fois la mise à niveau terminée, l'Assistant se ferme et le nouveau projet apparaît dans l'**Explorateur de solutions**.

Pour afficher les résultats de la mise à niveau

1. Dans l'**Explorateur de solutions**, sélectionnez **_UpgradeReport.htm** pour ouvrir le rapport de mise à niveau.

   Remarquez qu'aucun problème général devant être traité n'est spécifié dans le rapport ; pour PicForm.vb, le rapport répertorie six erreurs et aucun avertissement.

   Remarque

   Si le rapport contient des avertissements, ceux-ci figurent également dans la fenêtre **Liste des tâches** en tant qu'éléments UPGRADE_WARNING. Les avertissements concernent le code pouvant provoquer de légères différences dans le comportement à l'exécution de votre application. Dans la fenêtre **Liste des tâches**, double-cliquez sur un élément UPGRADE_WARNING pour atteindre directement le code que vous souhaitez modifier.

2. Dans la colonne **Nouveau nom de fichier**, cliquez sur le signe plus pour développer la section **PicForm.vb**. Cette opération permet d'afficher une liste détaillée des problèmes de mise à niveau concernant le formulaire.

   Remarquez que les problèmes s'appliquent à trois emplacements différents : la procédure **ClearPictureBox_Click**, la procédure **Timer1_Timer** et la disposition de formulaire. Pour afficher une rubrique d'aide qui explique chaque problème, cliquez sur le lien **Description**. Dans notre exemple, les problèmes de présentation ne nécessitent aucune intervention ; la rubrique d'aide révèle que deux des propriétés du contrôle **PictureBox** Visual Basic 6.0 n'ont pas leur équivalent dans Visual Basic 2005.

Pour corriger l'erreur liée à ClearPictureBox_Click

1. Dans l'**Explorateur de solutions**, sélectionnez **PicForm.vb**. Dans le menu **Affichage**, choisissez **Code**.
2. Dans l'éditeur de code, sélectionnez la procédure **ClearPictureBox_Click**.

> **Conseil**
>
> Un commentaire UPGRADE_ISSUE a été ajouté à la procédure. À la fin du document, vous pouvez cliquer sur le lien qui affiche la rubrique d'aide associée. Tous les problèmes (mis à part les problèmes de présentation) sont ajoutés en tant que commentaires de code au cours d'une mise à niveau, y compris les commentaires UPGRADE_NOTE expliquant ce q ui s'est passé au cours de la mise à niveau ; ces commentaires n'apparaissent pas dans le rapport de mise à niveau.

3. Ajoutez le code suivant à la procédure **ClearPictureBox_Click** :

   VB

```
Dim g As Graphics = Picture1.CreateGraphics()
g.Clear(Picture1.BackColor)
g.Dispose()
```

   L'unique erreur dans la procédure d'origine est l'appel à la méthode **Picture1.Cls()**. Dans Visual Basic 2005, les contrôles intrinsèques ne fournissent pas d'accès direct à leurs surfaces de dessin à l'aide de méthodes (comme dans Visual Basic 6.0). Toutes les opérations graphiques sont gérées avec un objet spécial de type **Graphics**. Vous pouvez accéder à la surface de dessin d'un contrôle en appelant la méthode **CreateGraphics()** pour obtenir une instance d'un objet **Graphics**.

4. Supprimez ou mettez en commentaire la ligne d'origine : `Picture1.Cls()`.

Pour corriger les erreurs liées à Timer1_Timer

1. Dans l'éditeur de code, sélectionnez la procédure **Timer1_Tick**.

> **Remarque**
>
> L'événement **Timer** du contrôle Visual Basic 6.0 **Timer** est mis à niveau en événement **Tick** du composant Visual Basic 2005 **Timer**. Le rapport de mise à niveau affiche le nom de l'ancien événement alors que l'éditeur de code utilise le no m du nouvel événement.

2. Au démarrage de la fonction, ajoutez le code suivant :

   VB

```
Dim g As Graphics = Picture1.CreateGraphics()
```

   Comme dans la procédure précédente, les erreurs commencent par l'absence d'objet **Graphics**.

3. Recherchez le premier UPGRADE_ISSUE, à savoir la méthode **Picture1.Print**. Ajoutez le code ci-dessous :

   VB

```
g.DrawString("VB .NET ROCKS!", Me.Font, New SolidBrush( _
  Color.Yellow), 0, 0)
```

   La méthode **DrawString** de l'objet **Graphics** remplace la méthode Visual Basic 6.0 **Print**. Là où la méthode **Print** prenait un seul argument **Text**, **DrawString** prend également des arguments qui spécifient **Font**, un objet **Brush** qui spécifie un objet **Color**, et les coordonnées de départ pour dessiner le texte.

4. Supprimez ou mettez en commentaire la ligne d'origine : `Picture1.Print("Visual Basic ROCKS!")`.

5. Recherchez l'élément UPGRADE_ISSUE suivant, **Picture1.Cls()**. Remplacez le code par la ligne suivante :

   VB

   ```
   g.Clear(Picture1.BackColor)
   ```

6. Recherchez le dernier élément UPGRADE_ISSUE, **Picture1.PaintPicture**. Ajoutez le code ci-dessous :

   VB

   ```
   g.DrawImage(sourceImage.Image, CSng(LeftPos * Picture1.Size.Width), _
     0)
   ```

   Dans ce cas, la méthode **DrawImage** de la classe **Graphics** remplace la méthode **PaintPicture**.

7. Supprimez ou mettez en commentaire la ligne d'origine : `Picture1.PaintPicture(sourceImage, LeftPos * VB6.PixelsToTwipsX(Picture1.Width), 0)`.

8. À la fin de la procédure, ajoutez le code suivant :

   VB

   ```
   g.Dispose()
   ```

   Vous devez utiliser la méthode **Dispose** pour libérer les ressources mémoire associées à l'objet **Graphics**.

Pour tester l'application

1. Dans le menu **Déboguer**, cliquez sur **Démarrer**.

Remarque
si vous êtes invités à enregistrer le fichier solution, cliquez sur **Enregistrer** pour enregistrer le fichier et exécuter l'appli cation.

2. Cliquez sur le bouton **ShowImage**.

   Vous remarquerez que l'image scintille lorsqu'elle se dessine. Ce problème provient du fait que c'est l'intégralité de la zone qui est effacée avant que chaque image mise à jour se dessine à l'écran.

Modification avancée de l'application

La solution consiste ici à ne dessiner que sur les parties nécessaires de l'écran, au lieu de la zone entière représentée par l'objet **Graphics**. Dans ce cas, vous voudrez vous assurer que, lorsque l'image se dessine à l'écran, tout pixel, encore à l'écran, correspondant au dernier appel à **DrawImage()** se confond avec la couleur de l'arrière-plan. Comme l'image se déplace de la droite vers la gauche, il vous suffit d'obscurcir les pixels supplémentaires situés à droite de l'image.

Pour résoudre le problème de scintillement de l'écran

1. Dans la procédure **Timer1_Tick**, remplacez la méthode `g.Clear(Picture1.BackColor)` par le code suivant :

   VB

   ```
   Dim imageWidth As Integer = sourceImage.Image.Width
   Dim imageHeight As Integer = sourceImage.Image.Height
   Dim left As Double = imageWidth + (Picture1.Size.Width * LeftPos)

   g.FillRectangle(New SolidBrush(Me.BackColor), _
     New Rectangle(left, 0, 6, imageHeight))
   ```

À la place de la méthode **Clear**, utilisez la méthode **FillRectangle** pour obscurcir, à l'aide de la couleur de l'arrière-plan, les pixels restants situés à droite de l'image qui se dessine. Pour ce faire, calculez d'abord la coordonnée la plus à gauche du rectangle. Cette opération est effectuée par les trois premières lignes du code.

2. Dans le menu **Déboguer**, cliquez sur **Démarrer**.
3. Cliquez sur le bouton **ShowImage**.

   L'image se déplace alors sans scintiller.

Il reste encore un problème à résoudre. Vous avez probablement remarqué que la police du message affiché dans le contrôle **PictureBox** est plus petite que celle de la version d'origine. L'explication est la suivante : dans l'application Visual Basic 6.0, la propriété **Font** du contrôle **PictureBox** était définie en Arial Bold 16 au moment du design. Il n'y a pas de propriété **Font** pour le contrôle Visual Basic 2005 **PictureBox**. À la place, la police par défaut du formulaire (`Me.Font`) est utilisée. Vous pouvez y remédier en déclarant un nouvel objet **Font**.

Pour modifier la police

1. Dans la procédure **Timer1_Tick**, ajoutez la déclaration suivante :

   VB

```
Dim f As System.Drawing.Font = New System.Drawing.Font("Arial", _
  16, FontStyle.Bold)
```

2. Dans l'appel à `g.DrawString`, remplacez le paramètre de police `Me.Font` par `f`.
3. Dans le menu **Déboguer**, cliquez sur **Démarrer**.
4. Cliquez sur le bouton **ShowImage**.

   À présent, le texte s'affiche dans la police correcte.

Maintenant que l'application équivaut à l'application Visual Basic 6.0 d'origine, vous pouvez ajouter du code pour l'améliorer. Après tout, il est inutile de mettre à niveau l'application si vous ne souhaitez pas tirer parti de Visual Basic 2005.

L'étape suivante vous permet d'ajouter des fonctionnalités pour obtenir une fermeture en fondu du formulaire lors de sa fermeture.

Remarque

L'effet de fondu n'est pas visible sur des écrans 256 couleurs ; pour un résultat optimal, vous devez attribuer à la profondeur chromatique la valeur 65536 couleurs ou Couleurs vraies 24 bits. La propriété **Opacity** fonctionne uniquement sur Windows XP.

Pour améliorer l'application

1. Dans l'éditeur de code, sélectionnez la liste déroulante **Nom de la classe** et choisissez **(Form1 Events)**.
2. Dans la liste déroulante **Nom de la méthode**, sélectionnez l'événement **FormClosing**.
3. Dans la procédure **Form1_FormClosing**, ajoutez le code suivant :

   VB

```
Dim i As Single
For i = 1 To 0 Step -0.1
  Me.Opacity = i
  Application.DoEvents()
  System.Threading.Thread.Sleep(100)
Next
```

4. Dans le menu **Déboguer**, cliquez sur **Démarrer**.

5. Fermez le formulaire et observez la façon dont il disparaît de l'écran.

   Vous pouvez contrôler la vitesse de fermeture en fondu en augmentant la taille de la variable Step (ou en réduisant le délai **Sleep**).

Voir aussi

Référence

Vue d'ensemble des Windows Forms

Concepts

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Autres ressources

Mise à niveau de Visual Basic 6.0

Référence de mise à niveau Visual Basic 6.0

Les rubriques de la section suivante de la Table des matières sont des rubriques de référence liées à l'**Assistant Mise à niveau** et à l'espace de noms **VisualBasic.Compatibility**.

Voir aussi

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Assistant Mise à niveau

Utilisez l'Assistant Mise à niveau pour mettre à niveau des projets Visual Basic 6.0 vers Visual Basic 2005. L'Assistant crée un nouveau projet, copie chaque fichier du projet d'origine dans le nouveau projet et le modifie si besoin est, puis génère un rapport indiquant ce qui a été fait et ce qui reste à faire pour terminer la mise à niveau.

Pour accéder à l'Assistant Mise à niveau, ouvrez un fichier projet Visual Basic 6.0 (.vbp) dans Visual Basic 2005.

Liste des éléments d'interface utilisateur

L'Assistant Mise à niveau inclut les options suivantes.

Vers quel type de projet voulez-vous mettre à niveau votre projet existant ?

Si le projet Visual Basic 6.0 est un projet EXE ActiveX ou EXE Document ActiveX, vous pouvez le mettre à niveau vers un fichier .exe ou .dll. Pour tous les autres types de projet, l'option appropriée est déjà sélectionnée et vous ne pouvez pas en choisir une autre.

Où souhaitez-vous créer votre nouveau projet ?

Spécifie l'emplacement du nouveau projet. Par défaut, le nouveau projet est placé dans un répertoire situé juste au-dessous du projet Visual Basic 6.0. Le nouveau projet est appelé *ProjectName*.NET, où *ProjectName* désigne le nom du projet à mettre à niveau.

Voir aussi

Concepts

Assistant Mise à niveau de Visual Basic

Autres ressources

Mise à niveau d'applications créées dans des versions précédentes de Visual Basic

Référence d'espace de noms VisualBasic.Compatibility

Les fonctions et objets de l'espace de noms **VisualBasic.Compatibility** sont fournis pour être utilisés par les outils de mise à niveau de Visual Basic 6.0 à Visual Basic 2005. Dans la plupart des cas, ils dupliquent des fonctionnalités qui peuvent être obtenues à l'aide d'autres parties du .NET Framework. Ils sont nécessaires uniquement là où le modèle de code Visual Basic 6.0 diffère significativement de l'implémentation .NET.

Fonctions et objets dans l'espace de noms

Les tables suivantes répertorient les fonctions et les objets de l'espace de noms **VisualBasic.Compatibility**. D'autres fonctions et objets en rapport avec l'espace de noms **Microsoft.Compatibility.Data** ne sont pas répertoriés ; ils sont équivalents à la liaison de données ADO dans Visual Basic 6.0.

Fonctions de VisualBasic.Compatibility

Fonction	Description
CopyArray	Duplique la fonctionnalité de Visual Basic 6.0 permettant d'assigner un tableau à une variante.
CursorToIPicture	Utilisé pour la conversion de format.
Eqv	Duplique l'opérateur **Eqv** (équivalence) de Visual Basic 6.0.
FontChangeBold	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontChangeGdiCharSet	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontChangeItalic	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontChangeName	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontChangeSize	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontChangeStrikeout	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontChangeUnderline	Utilisé lors de la conversion de polices Visual Basic 6.0 en objets **Font** Visual Basic 2005.
FontToIFont	Utilisé pour la conversion de format.
Format	Duplique la fonction **Format** de Visual Basic 6.0.
FromPixelsX	Utilisé pour la conversion de coordonnées.
FromPixelsY	Utilisé pour la conversion de coordonnées.
FromPixelsUserHeight	Utilisé pour la conversion de coordonnées.
FromPixelsUserWidth	Utilisé pour la conversion de coordonnées.
FromPixelsUserX	Utilisé pour la conversion de coordonnées.
FromPixelsUserY	Utilisé pour la conversion de coordonnées.

GetActiveControl	Duplique la propriété **Screen.ActiveControl** de Visual Basic 6.0.
GetCancel	Duplique la fonctionnalité de la propriété **Cancel** d'un **CommandButton** de Visual Basic 6.0.
GetDefault	Duplique la fonctionnalité de la propriété **Default** d'un **CommandButton** de Visual Basic 6.0.
GetEXEName	Duplique la propriété **App.EXEName** de Visual Basic 6.0.
GetHInstance	Duplique la propriété **App.HInstance** de Visual Basic 6.0.
GetItemData	Duplique la fonctionnalité de la propriété **ItemData** d'un **ListBox** ou d'un **ComboBox** Visual Basic 6.0.
GetItemString	Duplique la fonctionnalité permettant de récupérer une valeur d'une liste dans un **ListBox** ou un **Combo Box** de Visual Basic 6.0.
GetPath	Duplique la propriété **App.Path** de Visual Basic 6.0.
IconToIpicture	Utilisé pour la conversion de format.
IFontToFont	Utilisé pour la conversion de format.
ImageToIPicture	Utilisé pour la conversion de format.
ImageToIPictureDisp	Utilisé pour la conversion de format.
Imp	Duplique l'opérateur **Imp** (implication) de Visual Basic 6.0.
IPictureDispToImage	Utilisé pour la conversion de format.
IPictureToImage	Utilisé pour la conversion de format.
LoadResData	Duplique la fonction **LoadResData** de Visual Basic 6.0.
LoadResPicture	Duplique la fonction **LoadResPicture** de Visual Basic 6.0.
LoadResString	Duplique la fonction **LoadResString** de Visual Basic 6.0.
PixelsToTwipsX	Utilisé pour la conversion de coordonnées.
PixelsToTwipsY	Utilisé pour la conversion de coordonnées.
SendKeys	Duplique la fonction **SendKeys** de Visual Basic 6.0.
SetCancel	Duplique la fonctionnalité de la propriété **Cancel** d'un **CommandButton** de Visual Basic 6.0.
SetDefault	Duplique la fonctionnalité de la propriété **Default** d'un **CommandButton** de Visual Basic 6.0.
SetItemData	Duplique la fonctionnalité de la propriété **ItemData** d'un **ListBox** ou d'un **ComboBox** Visual Basic 6.0.
SetItemString	Duplique la fonctionnalité permettant de définir une valeur d'une liste dans un **ListBox** ou un **ComboBox** de Visual Basic 6.0.

SetResourceBaseName	Utilisé avec les fonctions **LoadRes** lorsqu'un fichier de ressources utilise une convention d'affectation de nom non standard.
ShowForm	Duplique la fonctionnalité de la méthode **Show** d'un formulaire Visual Basic 6.0.
TabLayout	Utilisé lors de la conversion des instructions **Debug.Print** qui contiennent les mots clés **Tab** ou **Spc**.
ToPixelsX	Utilisé pour la conversion de coordonnées.
ToPixelsY	Utilisé pour la conversion de coordonnées.
ToPixelsUserHeight	Utilisé pour la conversion de coordonnées.
ToPixelsUserWidth	Utilisé pour la conversion de coordonnées.
ToPixelsUserX	Utilisé pour la conversion de coordonnées.
ToPixelsUserY	Utilisé pour la conversion de coordonnées.
TwipsPerPixelX	Duplique la propriété **Screen.TwipsPerPixelX** de Visual Basic.
TwipsPerPixelY	Duplique la propriété **Screen.TwipsPerPixelY** de Visual Basic.
TwipsToPixelsX	Utilisé pour la conversion de coordonnées.
TwipsToPixelsY	Utilisé pour la conversion de coordonnées.
ValidateControls	Duplique la fonctionnalité de la méthode **ValidateControls** d'un formulaire Visual Basic 6.0.
WhatsThisMode	Duplique la fonctionnalité de la méthode **WhatsThisMode** d'un formulaire Visual Basic 6.0.
Zorder	Utilisé pour la conversion de l'ordre de plan des contrôles.

Objets VisualBasic.Compatibility

Objet	Description
BaseControlArray	Classe parente pour l'émulation de tableau de contrôles Visual Basic 6.0.
BaseOcxArray	Classe parente pour les tableaux émulés des contrôles ActiveX.
ButtonArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **CommandButton**.
CheckBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **CheckBox**.
CheckedListBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **ListBox** avec la valeur Checked affectée à la propriété **Style**.
ComboBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **ComboBox**.
DirListBox	Émule un contrôle **DirListBox** Visual Basic 6.0.
DirListBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **DirListBox**.

DriveListBox	Émule un contrôle **DriveListBox** Visual Basic 6.0.
DriveListBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **DriveListBox**.
FileListBox	Émule un contrôle **FileListBox** Visual Basic 6.0.
FileListBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **FileListBox**.
FixedLengthString	Émule des chaînes de longueur fixe Visual Basic 6.0.
GroupBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **Frame**.
HScrollBarArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **HScrollBar**.
LabelArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **Label**.
ListBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **ListBox**.
ListBoxItem	Émule la propriété **ItemData** d'un contrôle **ListBox**.
MenuItemArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **Menu**.
PanelArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **PictureBox** qui contient des contrôles enfant s.
PictureBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **PictureBox**.
RadioButtonArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **OptionButton**.
TabControlArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **TabStrip**.
TextBoxArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **TextBox**.
TimerArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **Timer**.
VscrollBarArray	Émule un tableau de contrôles Visual Basic 6.0 de contrôles **VScrollBar**.

Voir aussi
Concepts
Bibliothèque de compatibilité Visual Basic 6.0
Autres ressources
Aide pour les utilisateurs de Visual Basic 6.0

Développement d'applications avec Visual Basic

Cette section décrit la documentation conceptuelle relative au langage Visual Basic.

Dans cette section

Utilisation de Visual Basic Editor

Contient l'aide sur les fonctionnalités de l'éditeur Visual Basic, telles que **Extraits de code IntelliSense**, **Propriétés du projet**, et sur la création de la documentation XML pour Visual Basic.

Sécurité et développement Visual Basic

Décrit les trois zones courantes où vous serez confronté à des problèmes de sécurité.

Programmation dans Visual Basic

Aborde divers sujets relatifs à la programmation.

Développement avec My

Présente une nouvelle fonctionnalité appelée **My**, qui donne accès à des informations et des instances d'objet par défaut associées à une application et son environnement d'exécution.

Accès aux données dans les applications Visual Basic

Contient de l'aide relative à l'accès aux données dans Visual Basic.

Débogage de l'application Visual Basic

Fournit des pointeurs vers la documentation relative aux fonctionnalités de débogage intégrées dans Visual Studio.

Gestion des exceptions et des erreurs en Visual Basic

Contient de l'aide relative à la gestion structurée et non structurée des exceptions, ainsi qu'à la configuration d'avertissements, à la fonctionnalité **Correction automatique de la compilation intelligente** et aux types d'erreurs.

Déploiement de l'application

Offre une vue d'ensemble des fonctionnalités de déploiement dans le .NET Framework et Visual Studio.

Création et utilisation de composants en Visual Basic

Définit le terme *composant* et explique quand et comment créer des composants.

Fonctionnalités plus poussées de Visual Basic

Offre des vues d'ensemble de différentes sujets, notamment les formulaires, les opérations de réseau et les services Web.

Personnalisation des projets et extension de My avec Visual Basic

Décrit comment personnaliser des modèles de projet pour fournir des objets **My** supplémentaires.

Rubriques connexes

Guide de programmation Visual Basic

Passe en revue les éléments essentiels de la programmation avec Visual Basic.

Référence Visual Basic

Contient la documentation de référence pour le langage Visual Basic.

Utilisation de Visual Basic Editor

En plus des fonctionnalités d'interface utilisateur principales que fournit Visual Studio, Visual Basic en ajoute quelques-unes qui contribuent à la productivité et à la facilité d'utilisation.

Dans cette section

Paramètres de Visual Basic

Explique comment optimiser votre expérience de l'environnement de développement.

Extraits de code IntelliSense Visual Basic

Présente une bibliothèque de code à laquelle vous pouvez accéder à partir de l'environnement de développement.

Comment : modifier des attributs de code

Explique comment utiliser la fenêtre **Propriétés** afin d'appliquer et de modifier des attributs pour vos types et membres.

Renommer, boîte de dialogue (Visual Basic)

Décrit le mécanisme d'attribution d'un nouveau nom aux identificateurs dans votre code.

Touches de raccourci par défaut de Visual Basic 6.0

Fournit des liens vers différentes pages répertoriant les touches de raccourci.

Aide destinée aux gestionnaires d'événements dans le code Visual Basic

Montre comment obtenir l'aide (F1) sur un gestionnaire d'événements.

Comment : compiler et exécuter un projet dans Visual Basic

Décrit vos options pour la compilation et l'exécution à partir de l'environnement de développement.

Référence

Modifier & Continuer (Visual Basic)

Décrit la fonctionnalité de débogage qui vous permet d'apporter des modifications à votre code pendant son exécution en mode arrêt.

Correction automatique de la compilation intelligente

Décrit de quelle manière cette fonctionnalité propose des corrections en cas d'erreur, ce qui vous permet de choisir une solution à appliquer au code.

Configuration d'avertissements en Visual Basic

Décrit comment désactiver les avertissements ou faire en sorte qu'ils soient traités comme des messages d'erreur.

Paramètres de Visual Basic

Les paramètres Visual Basic sont conçus pour optimiser l'interaction de l'utilisateur et maximiser la productivité des développeurs Visual Basic. Ils font partie intégrante de la fonctionnalité Paramètres Visual Studio qui permet de personnaliser et d'enregistrer les paramètres de votre environnement de développement intégré (IDE) Visual Studio.

Vous pouvez également porter ou recharger vos paramètres IDE sur d'autres ordinateurs. Pour plus d'informations, consultez Comment : partager des paramètres entre ordinateurs.

Fenêtres et affichages

Fonctionnalité	**Affiché par défaut ?**	**Remarques**
Affichage de classes	Non	Masque les **membres et les types masqués**
Commande, fenêtre	Non	
Fenêtre **Aide dynamique**	Non	Ne s'affiche pas lorsque vous appuyez sur F1.
Fenêtre Exécution	Oui, lors du démarrage du débogage.	
Explorateur d'objets	Non	S'affiche, par défaut : • Espaces de noms • Membres publics • Membres hérités
Sortie, fenêtre	Oui, au démarrage de la génération.	
Propriétés, fenêtre	Oui	
Explorateur de solutions	Oui	
Explorateur de serveurs/Explorateur de bases de données	Non	Affiche les connexions de données et les services système disponibles.
Page de démarrage	Oui	Apparaît au premier démarrage de l'IDE. Vous pouvez la personnaliser.
Liste des tâches (Visual Studio)	Oui, en présence d'avertissements ou d'erreurs de compilation.	
Boîte à outils	Oui	Affiche les contrôles et composants par ordre alphabétique.

De plus, les éléments suivants présentent certains comportements activés lorsque vous sélectionnez **Paramètres de développement Visual Basic**..

Boîtes de dialogue

Fonctionnalité	**Comportement**
Nouveau projet, boîte de dialogue	Lorsque d'autres plates-formes sont installées sur l'ordinateur, l'IDE active la zone de liste déroulante qui vous permet de spécifier la plate-forme ciblée.

Options, boîte de dialogue (Visual Studio)	Visual Basic active des pages **Options** simplifiées. Cliquez sur la case à cocher **Afficher tous les paramètres** pour afficher la liste complète des options.

Clavier

Fonctionnalité	Comportement
Touches de raccourci par défaut de Visual Basic 6.0	Prend en charge le mappage de clavier Visual Basic 6 typique.

Divers éléments IDE

Fonctionnalité	Comportement
Info-bulles	• Apparaissent pour tous les éléments de menu et les boutons de barre d'outils lorsque le pointeur de souris se positionne sur ceux-ci.
Extraits de code IntelliSense Visual Basic	Inclut une bibliothèque des extraits de code prêts à être insérés dans votre application.
Options IntelliSense spécifiques à Visual Basic	• Fournit toutes les fonctionnalités IntelliSense avec certaines améliorations de la productivité pour les développeurs Visual Basic.

Voir aussi

Tâches

Comment : personnaliser l'aide dynamique

Comment : modifier les paramètres de sélection

Comment : spécifier les paramètres pour une équipe

Référence

Importation et exportation de paramètres, Environnement, boîte de dialogue Options

Autres ressources

Assistant Importation et exportation de paramètres

Extraits de code IntelliSense Visual Basic

Visual Basic inclut une bibliothèque de code composée de plusieurs centaines de segments de code, appelés extraits de code IntelliSense, qui sont prêts à être insérés dans votre application. Chaque extrait de code effectue une tâche de programmation complète, par exemple créer une exception personnalisée, envoyer un message électronique ou dessiner un cercle. Quelques clics de souris suffisent pour insérer un extrait dans votre code source. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic). Vous pouvez également créer vos propres extraits de code en fonction des besoins de votre entreprise, les ajouter à la bibliothèque, puis les utiliser lorsque cela est nécessaire. Pour plus d'informations, consultez Création et utilisation d'extraits de code IntelliSense.

Un outil de productivité pour les utilisateurs de Visual Basic

Ces extraits de code augmentent votre productivité en réduisant le temps passé à rechercher des exemples de code, en réduisant le temps nécessaire pour apprendre à utiliser une fonctionnalité peu familière et en prenant en charge la réutilisation du code.

La bibliothèque de code prend en charge les fonctionnalités suivantes :

- Insérer l'un des extraits de code dans l'**Éditeur de code**.
- Créer de nouvelles tâches pouvant être réutilisées dans vos projets.
- Créer de nouvelles tâches à partager avec votre groupe de travail et vos collègues.
- Modifier les tâches.
- Télécharger d'autres tâches auprès de fournisseurs tiers.

Ces blocs de code sont disponibles partout dans Visual Studio :

- Vous pouvez les ajouter via le menu contextuel de l'**éditeur de code**.
- Vous pouvez faire glisser les fichiers de code XML depuis l'**Explorateur Windows** vers votre fichier de code source.

Voir aussi

Tâches

Comment : modifier des extraits de code existants
Comment : partager des extraits de code avec d'autres développeurs
Comment : supprimer un extrait de code ajouté à votre code
Comment : affecter un nom de raccourci à un extrait de code
Dépannage des extraits
Comment : gérer des extraits de code
Comment : rechercher des extraits de code en ligne

Concepts

Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense
Considérations relatives à la sécurité lors de l'utilisation d'extraits de code

Comment : insérer des extraits de code dans votre code (Visual Basic)

Vous pouvez insérer des extraits de code IntelliSense à l'aide des outils de navigation dans l'**éditeur de code** ou par un glisser-déplacer à partir de l'**Explorateur Windows**. Lorsque vous connaissez le nom de l'extrait de code à insérer, vous pouvez simplement taper son raccourci et appuyer sur la touche TABULATION. Lorsque vous devez rechercher un extrait de code spécifique, vous pouvez utiliser le sélecteur d'extrait de code afin d'obtenir des listes classées d'extraits de code.

Après avoir inséré l'extrait de code, vous pouvez le modifier pour travailler dans votre application. Par exemple, il est possible qu'un extrait de code particulier insère une nouvelle méthode dans votre application. Si vous avez inséré cette nouvelle méthode dans une méthode existante, vous devez supprimer quelques lignes supplémentaires de code pour résoudre les erreurs de compilation. Vous pouvez également modifier les messages destinés aux utilisateurs ou les noms de fichiers trouvés dans l'extrait de code. Pour plus d'informations, consultez
Comment : personnaliser le code d'un extrait une fois qu'il est inséré dans votre code.

Pour insérer un extrait de code en parcourant l'éditeur de code

1. Cliquez avec le bouton droit sur l'**éditeur de code** à l'endroit où vous souhaitez insérer le code.
2. Cliquez sur **Insérer un extrait** dans le menu contextuel. Le sélecteur d'extrait de code IntelliSense apparaît.

   À la place des étapes 1 et 2, vous pouvez taper un point d'interrogation et appuyer sur la touche TABULATION dans l'**éditeur de code** pour ouvrir le sélecteur d'extrait de code.
3. Naviguez jusqu'à la tâche de votre choix et cliquez dessus. L'extrait de code est inséré dans votre code.

   Après avoir ajouté l'extrait à votre code, certaines parties peuvent nécessiter une personnalisation, par exemple remplacer les noms de variables par des noms plus appropriés. L'éditeur peut vous aider à effectuer ces opérations. Pour plus d'informations, consultez Comment : personnaliser le code d'un extrait une fois qu'il est inséré dans votre code.

   Les extraits de code qui apparaissent dans le sélecteur sont stockés en tant que fichiers dans des sous-dossiers à des emplacements de fichiers spécifiques. Les emplacements de fichiers peuvent être vérifiés et modifiés à l'aide du Gestionnaire des extraits de code. Il existe un fichier pour chaque extrait de code, et l'extension de nom pour ces fichiers est .snippet. Les extraits de code créés ou obtenus à partir d'une autre source peuvent être stockés n'importe où dans le système de fichiers ; ils n'apparaîtront toutefois pas dans le menu contextuel sauf s'ils résident dans l'un des dossiers spécifiés. Pour plus d'informations, consultez le gestionnaire d'extrait de code.

Pour insérer un extrait de code lorsque vous connaissez son raccourci

- Tapez le raccourci de l'extrait dans l'**éditeur de code** et appuyez sur la touche TABULATION. L'extrait de code sera inséré.

  - ou -

- Tapez les premiers caractères du raccourci de l'extrait dans l'**éditeur de code**, suivi d'un point d'interrogation, et appuyez sur la touche TABULATION. Un sélecteur apparaîtra avec la liste des extraits dont les raccourcis commencent par les caractères tapés. En parcourant la liste, vous verrez le titre du raccourci sélectionné apparaître dans une Info-bulle.

Il existe deux méthodes pour rechercher le nom du raccourci pour un extrait de code :

- Utilisez le Gestionnaire des extraits de code pour consulter les raccourcis correspondants à tous les extraits de code.
- Le raccourci est affiché dans l'Info-bulle de description dans le sélecteur d'extrait de code.

Pour insérer un extrait dans votre code à l'aide de l'Explorateur Windows

1. Ouvrez l'**Explorateur Windows**.
2. Naviguez dans l'**Explorateur Windows** jusqu'au dossier contenant le fichier à insérer. Si vous avez choisi l'emplacement par défaut pour l'installation de Visual Studio 2005, les extraits sont stockés dans C:\Program Files\Microsoft Visual Studio 8\Vb. Si vous avez choisi l'emplacement par défaut pour l'installation de Visual Basic 2005 Express, les extraits sont stockés dans C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\VBExpress\Snippets\1033.

3. Faites glisser le fichier depuis l'**Explorateur Windows** vers l'emplacement voulu dans votre code.

Voir aussi

Tâches

Comment : rechercher des extraits de code en ligne

Référence

Création et utilisation d'extraits de code IntelliSense

Concepts

Extraits de code IntelliSense Visual Basic

Considérations relatives à la sécurité lors de l'utilisation d'extraits de code

Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense

Comment : personnaliser le code d'un extrait une fois qu'il est inséré dans votre code

Lorsque vous ajoutez un extrait à votre code, un ou plusieurs points de remplacement sont mis en surbrillance dans le code de l'extrait inséré. Vous pouvez choisir de modifier chaque point de remplacement ou non. Si vous positionnez le pointeur de votre souris sur le point de remplacement, une Info-bulle s'affiche pour vous expliquer comment modifier le code. L'**éditeur de code** reconnaît un extrait de code comme une unité distincte de votre code avoisinant jusqu'à ce que vous fermiez le fichier de code source. Ainsi, les points de remplacement restent en surbrillance jusqu'à la fermeture du fichier source.

Pour personnaliser le code inséré pour votre application

1. Utilisez la touche TABULATION pour passer d'un point de remplacement à un autre. Utilisez les touches MAJ+TAB pour revenir à un point de remplacement précédent.
2. Appuyez sur CTRL+ESPACE pour appeler IntelliSense.
3. Sélectionnez un élément dans la liste ou tapez le remplacement de votre choix.
4. S'il y a encore des erreurs de compilation dans le code, vous devrez peut-être apporter d'autres modifications à votre code source. Par exemple, si la tâche consiste à rechercher un nœud dans un contrôle TreeView nommé `TreeView1`, alors vous devrez ajouter à votre formulaire un contrôle **TreeView** nommé `TreeView1`.

Voir aussi

Tâches

Comment : insérer des extraits de code dans votre code (Visual Basic)

Référence

Création et utilisation d'extraits de code IntelliSense

Concepts

Extraits de code IntelliSense Visual Basic

Considérations relatives à la sécurité lors de l'utilisation d'extraits de code

Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense

Considérations relatives à la sécurité lors de l'utilisation d'extraits de code

Les extraits de code IntelliSense installés par Visual Studio ne constituent pas eux-mêmes un risque de sécurité. Selon l'endroit où ils sont utilisés et la façon dont ils sont modifiés, ils peuvent créer des problèmes de sécurité dans votre application. Les extraits de code téléchargés à partir d'Internet doivent être traités comme n'importe quel autre contenu téléchargé.

Extraits de code provenant d'Internet

Lorsque vous téléchargez des fichiers d'extraits de code à partir d'Internet, vous devez prendre en considération les éléments suivants :

- Une extension de fichier .snippet ne garantit pas que le fichier contient du texte XML brut. Pour plus de sécurité, effectuez vos téléchargements à partir de sites de confiance et utilisez un logiciel anti-virus à jour.
- Il peut arriver que l'URL de l'aide qui fait partie du fichier d'extrait de code exécute un fichier script nuisible ou affiche un site Web offensif. Cette URL n'est pas affichée dans l'**éditeur de code** lorsque vous insérez l'extrait de code, mais elle est visible si vous modifiez le fichier d'extrait dans l'Éditeur XML ou dans un autre éditeur tel que le Bloc-notes.
- Le code lui-même pourrait contenir du code susceptible d'endommager votre système si vous l'exécutez. Examinez soigneusement la source avant d'exécuter le code. Certaines parties du code peuvent se trouver dans des sections #Region, directive réduites. Développez ces sections pour lire le code.
- L'extrait de code peut contenir des références. Ces références sont ajoutées de manière silencieuse à votre projet et peuvent être chargées à partir de n'importe quel emplacement sur votre système. Ces références ont pu être téléchargées sur votre ordinateur depuis l'emplacement où vous avez téléchargé l'extrait de code. L'extrait de code peut ensuite appeler une méthode dans la référence qui exécute le code nuisible. Pour vous protéger contre une telle attaque, lisez avec soin le fichier d'extrait de code avant d'insérer l'extrait et effectuez vos téléchargements à partir de sites de confiance.

Sécurité pour tous les extraits de code

Le niveau de sécurité d'un extrait de code dépend de l'emplacement où il est utilisé dans le code source et de la façon dont il est modifié après avoir été inséré dans le code. La liste suivante contient quelques-unes des zones qui doivent faire l'objet d'une attention particulière :

- Accès aux fichiers et aux bases de données
- Sécurité d'accès du code
- Protection des ressources (par exemple les journaux des événements, le Registre)
- Stockage des informations confidentielles
- Vérification des entrées
- Passages des données aux technologies de script

Pour plus d'informations, consultez Sécurisation des applications.

Voir aussi

Tâches

Comment : rechercher des extraits de code en ligne

Référence

Création et utilisation d'extraits de code IntelliSense

Concepts

Sécurisation des applications

Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense

Le code dans chaque extrait de code IntelliSense affiche uniquement une méthode de base pour accomplir une tâche. Pour la plupart des applications, le code devra être modifié en fonction de l'application. Cette rubrique décrit plusieurs modifications courantes qui doivent être apportées au code.

Gestion des exceptions

En général, si le code contient un bloc **Try...Catch**, il interceptera et lèvera une nouvelle fois toutes les exceptions. Ce n'est pas forcément la meilleure solution pour votre projet. Pour chaque exception, il existe plusieurs façons de répondre. Voici quelques actions possibles :

- Ajoutez du code à chaque bloc **Catch** pour l'action choisie.
- Ne faites rien, en supprimant le code qui apparaît dans le bloc **Catch**. Dans ce cas, l'application ignore l'erreur. Il est peu probable que votre application puisse récupérer normalement si vous choisissez cette solution.
- Récupérez plus d'entrées de la part de l'utilisateur pour essayer de résoudre la condition.
- Pendant l'exécution du bloc **Try**, annulez toutes les modifications apportées aux données d'application avant que l'exception n'ait été levée.
- Répercutez l'exception vers la méthode d'appel.
- Levez une exception que vous définissez pour cette application.
- Supprimez l'instruction **Catch** pour cette exception et laissez la méthode d'appel la gérer. Cette solution est particulièrement appropriée si l'exception ne concerne pas votre tâche.
- Ajoutez des blocs **Catch** supplémentaires au code pour intercepter des types spécifiques d'exceptions que vous souhaitez gérer.
- Ajoutez un bloc **Finally** pour ajouter du code à exécuter après les blocs **Try** et **Catch**.

Remplacement des chaînes

Si le code contient une valeur de chaîne, il s'agit en général d'une chaîne spécifique, telle que "c:\filename.txt". Cette chaîne codée de manière irréversible est utile à des fins de démonstration, mais elle n'est probablement pas la chaîne correcte pour votre application. Voici des sources potentielles de chaînes pour votre application :

- Une variable chaîne.
- Une méthode ou une propriété qui retourne une chaîne, telle que la fonction **InputBox**.
- Une chaîne d'un contrôle Windows Forms, par exemple TextBox ou ComboBox.

Emplacements des fichiers

D'après les noms affichés dans le code, la plupart des fichiers se situent soit dans **Mes documents**, soit dans c:\. Lorsque vous changez l'emplacement des fichiers, vous devez notamment prendre en compte les éléments suivants :

- Trouver un emplacement accessible. Les utilisateurs n'ont peut-être pas accès au dossier **\Program Files** de l'ordinateur, donc stocker les fichiers avec les fichiers d'application n'est pas la solution appropriée.
- Trouver un emplacement sécurisé. Le stockage des fichiers dans le dossier racine (c: \) ne constitue pas une solution sécurisée. Pour les données d'application, nous vous recommandons d'utiliser le dossier **\Application Data**. Pour les données utilisateur, l'application peut créer un fichier pour chaque utilisateur dans le dossier **\Mes documents**.
- Utiliser un nom de fichier valide. Vous pouvez utiliser le OpenFileDialog, composant (Windows Forms) et le SaveFileDialog, composant (Windows Forms) pour réduire les risques de choisir des noms de fichiers non valides. Gardez à l'esprit qu'entre le moment où l'utilisateur sélectionne un fichier et le moment où votre code manipule le fichier, il se peut que le fichier ait été supprimé. En outre, l'utilisateur ne dispose peut-être pas des autorisations pour écrire dans le fichier.

Contrôles et composants

Les contrôles et les composants référencés dans le code ont habituellement un nom attribué par défaut par le concepteur, tel que `Button1` ou `TreeView1`. Ces noms mettent en évidence le type de contrôle indiqué, mais ne correspondent peut-être pas aux noms que vous utilisez dans votre application.

Code manquant

Certains extraits de code affichent seulement une structure de langage vide, par exemple un bloc **Try...Catch**. Un autre scénario courant est lorsqu'une variable de validation, par exemple **IsValid**, a par défaut la valeur **True**. Le code fourni pourra être compilé et exécuté, mais il n'aura aucune fonctionnalité. Dans ces exemples, vous devrez ajouter le code correspondant à votre tâche.

Sécurité

Le niveau de sécurité d'un extrait de code dépend de l'emplacement où il est utilisé dans le code source et de la façon dont il est modifié après avoir été inséré dans le code. La liste suivante contient quelques-unes des zones qui doivent faire l'objet d'une attention particulière :

- Accès aux fichiers et aux bases de données
- Sécurité d'accès du code
- Protection des ressources (par exemple les journaux des événements, le Registre)
- Stockage des informations confidentielles
- Vérification des entrées
- Passages des données aux technologies de script

Pour plus d'informations, consultez Considérations relatives à la sécurité lors de l'utilisation d'extraits de code et Sécurisation des applications.

Voir aussi

Tâches

Comment : insérer des extraits de code dans votre code (Visual Basic)

Référence

Création et utilisation d'extraits de code IntelliSense

Concepts

Extraits de code IntelliSense Visual Basic

Considérations relatives à la sécurité lors de l'utilisation d'extraits de code

Sécurisation des applications

Dépannage des extraits

Les problèmes liés aux extraits de code IntelliSense sont généralement provoqués par deux problèmes : lorsque le fichier d'extrait est corrompu ou lorsque le contenu de ce fichier est mauvais.

Problèmes courants

Il est impossible de faire glisser l'extrait de code depuis l'Explorateur Windows vers un fichier source Visual Studio

- Le XML dans le fichier d'extrait est peut-être corrompu. L'**Éditeur XML** dans Visual Studio permet de localiser des problèmes dans la structure XML.
- Le fichier d'extrait n'est peut-être pas conforme au schéma de l'extrait de code. L'**Éditeur XML** dans Visual Studio permet de localiser des problèmes dans la structure XML.

Certaines erreurs de compilation ne sont pas mises en surbrillance dans le code

- Une référence de projet est peut-être manquante. Examinez la documentation relative à l'extrait de code. Si la référence n'est pas trouvée sur l'ordinateur, vous devez l'installer. Lorsque vous insérez un extrait de code, toutes les références nécessaires doivent également être ajoutées au projet. Si les informations de référence manquent dans l'extrait de code, vous pouvez signaler cette erreur au créateur de l'extrait.
- Une variable n'est peut-être pas définie. Les variables non définies dans un extrait de code doivent être mises en surbrillance. Si tel n'est pas le cas, vous pouvez signaler cette erreur au créateur de l'extrait.

Voir aussi

Référence

Création et utilisation d'extraits de code IntelliSense

Comment : modifier des attributs de code

Lorsque vous êtes dans l'**éditeur de code** de Visual Basic, vous pouvez utiliser la fenêtre **Propriétés** pour appliquer et modifier les attributs de vos types et vos membres. Les attributs qui sont répertoriés dans la fenêtre **Propriétés** peuvent être appliqués en affectant à l'attribut la valeur **True** ou en affectant à la propriété **Apply** de l'attribut la valeur **True**. Après avoir appliqué les attributs, vous pouvez les modifier en modifiant les propriétés affichées dans la fenêtre **Propriétés**. Pour plus d'informations, consultez Attributs en Visual Basic.

Pour appliquer un nouvel attribut

1. Recherchez l'attribut souhaité dans la fenêtre **Propriétés**.
2. Si l'attribut souhaité n'est pas répertorié, vous devez appliquer manuellement l'attribut dans le code. Pour plus d'informations, consultez Application des attributs.
3. Effectuez l'une des opérations suivantes :
   - Si l'attribut a des propriétés, affectez à la propriété **Apply** la valeur **True**.

   - ou -

   - Si l'attribut n'a pas de propriétés, affectez à l'attribut la valeur **True**.

Voir aussi

Référence

Propriétés, fenêtre

Concepts

Vue d'ensemble des attributs dans Visual Basic
Utilisations courantes des attributs
Attributs utilisés dans Visual Basic
Application des attributs

Renommer, boîte de dialogue (Visual Basic)

Utilisez la boîte de dialogue **Renommer** prédéfinie de Visual Basic pour renommer des identificateurs de votre code pour les symboles, tels que les champs, les variables locales, les méthodes, les espaces de noms, les propriétés et les types. Vous pouvez ouvrir la boîte de dialogue **Renommer** en cliquant avec le bouton droit sur la déclaration d'élément.

Nouveau nom

Spécifie le nouveau nom de l'élément de code.

Emplacement

Identifie l'espace de noms à rechercher lors de l'exécution du changement de nom.

Changements de nom

La boîte de dialogue **Renommer** effectue des changements de nom légèrement différents, selon le type d'élément renommé.

Type d'élément	Changement de nom
Classe	Remplace toutes les déclarations et utilisations de la classe par le nouveau nom. Pour les classes partielles, le changement de nom est répercuté dans l'ensemble des éléments.
Champ	Remplace la déclaration et les utilisations du champ par le nouveau nom.
Variable locale	Remplace la déclaration et les utilisations de la variable par le nouveau nom.
Méthode	Remplace le nom de la méthode et toutes les références à cette méthode par le nouveau nom.
Espace de noms	Remplace le nom de l'espace de noms par le nouveau nom, dans la déclaration, dans toutes les instructions **Imports** et dans tous les noms complets.
Propriété	Remplace la déclaration et les utilisations de la propriété par le nouveau nom.

Refactorisation

Pour fournir une expérience de refactorisation plus complète, Visual Basic s'est associé à Developer Express Inc. pour obtenir la prise en charge de la refactorisation pour les développeurs. Consultez **Refactor!** (http://msdn.microsoft.com/vbasic/downloads/2005/tools/refactor/) sur le Centre de développement Visual Basic de MSDN pour obtenir plus de détails et d'instructions sur la façon de télécharger cet outil. **Refactor!** prend en charge plus de 15 fonctionnalités de refactorisation individuelles, telles que la réorganisation de paramètres, l'extraction de méthodes, l'encapsulation de champs et la création de surcharge.

Voir aussi

Tâches

Comment : renommer un identificateur

Autres ressources

Utilisation de Visual Basic Editor

Comment : renommer un identificateur

L'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio peut analyser votre code et utiliser les informations de portée pour vous aider à renommer les identificateurs. Cela fournit un meilleur contrôle de l'attribution de nouveaux noms que l'utilisation de la fonction de recherche et de remplacement.

Par exemple, prenez une classe qui définit deux sous-routines nommées `Test1` et `Test2`, chaque sous-routine définissant une variable nommée `Value`. Si vous remplacez le nom de la variable `Value` de `Test1` par `Counter`, tout le code dans `Test1` qui a référencé `Value` est mis à jour pour référencer `Counter`.Toutefois, le code dans `Test2` n'est pas modifié.

Pour renommer l'identificateur à partir d'une instance

1. Cliquez avec le bouton droit sur l'identificateur.
2. Choisissez **Renommer** dans le menu contextuel pour ouvrir la boîte de dialogue **Renommer**.
3. Remplacez le nom dans la zone **Nouveau nom** par le nouveau nom de l'identificateur.
4. Cliquez sur **OK** pour renommer toutes les références à l'identificateur.

Pour renommer l'identificateur à partir de sa déclaration

1. Sélectionnez le nom de l'identificateur à l'emplacement où il est déclaré.
2. Modifiez le nom de l'identificateur.

   Lorsque vous modifiez le nom, un symbole apparaît sous le côté droit du nom.
3. Positionnez le pointeur de votre souris sur le symbole pour activer la balise active.
4. Cliquez sur le glyphe de la balise active pour ouvrir le panneau.
5. Choisissez l'élément de panneau **Renommer** pour renommer les autres références à l'identificateur.

Voir aussi

Référence

Renommer, boîte de dialogue (Visual Basic)

Autres ressources

Utilisation de Visual Basic Editor

Touches de raccourci par défaut de Visual Basic 6.0

Les pages d'aide énumérées ci-dessous présentent les combinaisons de touches par défaut disponibles avec le schéma de clavier Visual Basic 6.0. Pour plus d'informations sur la modification des combinaisons par défaut, consultez Comment : utiliser les combinaisons de touches de raccourci.

Dans cette section

Touches de raccourci globales, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci communes, ainsi que les combinaisons de touches que vous pouvez utiliser à différents endroits dans l'IDE.

Touches de raccourci de l'aide intégrée, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lors de l'affichage des rubriques d'aide et de la navigation au sein de ces dernières.

Touches de raccourci de projet, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci que vous pouvez utiliser pour ajouter de nouveaux éléments à un projet, pour créer un projet, ou pour ouvrir des fichiers ou des projets.

Touches de raccourci du Concepteur HTML, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lorsque vous travaillez dans les modes **Design** et **HTML** du Concepteur HTML.

Touches de raccourci de l'Éditeur d'images, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lorsque vous travaillez dans l'éditeur d'image.

Touches de raccourci de débogage, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lorsque vous utilisez le débogueur.

Touches de raccourci de recherche et remplacement, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles avec les boîtes de dialogue **Rechercher**, **Remplacer**, **Rechercher dans les fichiers** et **Remplacer dans les fichiers**.

Touches de raccourci de Visual Database Tools, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lors de l'utilisation du Concepteur de requêtes ou du Concepteur de base de données.

Touches de raccourci de l'Explorateur d'objets, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lors de l'utilisation de l' **Explorateur d'objets**.

Touches de raccourci de manipulation des contrôles, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles pour déplacer et redimensionner des contrôles sur une aire de conception.

Touches de raccourci de navigation dans un texte, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lors d'un déplacement dans un code dans un document ouvert.

Touches de raccourci de sélection de texte, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lors de la sélection d'un texte dans un document ouvert.

Touches de raccourci de manipulation de texte, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles lors du déplacement, de la suppression ou du formatage d'un texte dans un document ouvert.

Touches de raccourci de la gestion des fenêtres, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci disponibles pour déplacer ou fermer des fenêtres Outil et des fenêtres

de document, ou pour y naviguer.

Touches de raccourci de macro, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci que vous pouvez utiliser avec des macros.

Touches de raccourci de la fenêtre Outil, option contextuelle par défaut de Visual Basic 6.0

Répertorie les combinaisons de touches de raccourci qui affichent les différentes fenêtres Outil.

Rubriques connexes

Comment : utiliser les combinaisons de touches de raccourci

Explique comment déterminer le schéma de configuration du clavier actuel et la combinaison de touches de raccourci pour une commande spécifique, et comment créer la méthode de mappage personnalisée et la nouvelle combinaison de touches de raccourci.

Touches de raccourci

Répertorie les schémas de configuration du clavier prédéfinis disponibles dans l'environnement de développement Visual Studio.

Touches de raccourci globales, option contextuelle par défaut de Visual Basic 6.0

Les combinaisons de touches de raccourci suivantes peuvent être utilisées à différents endroits dans l'environnement de développement intégré (IDE).

Commande	**Touches de raccourci**	**Description**
Edit.Copy	CTRL + C CTRL + INSER	Place un doublon de l'élément actuellement sélectionné dans le Presse-papiers du système.
Edit.Cut	CTRL + X MAJ + SUPPR	Supprime l'élément actuellement sélectionné dans le Presse-papiers du système.
Edit.CycleClipboardRing	CTRL + MAJ+ V CTRL + MAJ + INSER	Colle dans le fichier, à l'emplacement du point d'insertion, un élément de l'onglet **Presse-papiers circulaire** de la **Boîte à outils**, et sélectionne automatiquement l'élément collé. Vous pouvez afficher chaque élément du Presse-papiers en appuyant plusieurs fois sur les touches de raccourci.
Edit.Delete	SUPPR	Supprime le caractère de droite.
Edit.DeleteBackwards	RET.ARR MAJ + RET. ARR	Supprime le caractère de gauche.
Edit.GotoNextLocation	F12	Place le curseur sur l'élément suivant, tel qu'une tâche, dans la fenêtre **Liste des tâches** ou une occurrence dans la fenêtre **Résultats de la recherche**. Chaque fois que vous appuyez sur F12, le curseur passe à l'élément suivant de la liste.
Edit.GotoPreviousLocation	MAJ + F12	Place le curseur sur l'élément précédent dans la fenêtre **Liste des tâches** ou **Résultats de la recherche**.
Edit.ListMembers	CTRL + J	Répertorie les membres de la classe en cours pour compléter automatiquement les instructions lors de la modification du code.
Edit.ParameterInfo	CTRL + MAJ + I	Affiche une info-bulle qui contient des informations sur le paramètre en cours, en fonction du langage actuel.
Edit.Paste	CTRL + V MAJ + INSER	Insère le contenu du Presse-papiers à l'emplacement du point d'insertion.
Edit.QuickInfo	CTRL + I	Affiche la déclaration complète pour l'identificateur le plus proche du point d'insertion dans une info-bulle.

Edit.Redo	CTRL + MAJ + Z MAJ + ALT + RET. ARRIÈRE	Restaure l'action précédemment annulée.
Edit.SelectionCancel	ÉCHAP	Ferme un menu ou une boîte de dialogue, annule une opération en cours ou met le focus dans la fenêtre de document active.
Edit.Undo	ALT + RET. ARR CTRL + Z	Annule la dernière modification.
File.AddExistingItem	CTRL + D	Affiche la boîte de dialogue **Ajouter un élément existant**, dans laquelle vous pouvez ajouter un fichier existant au projet en cours.
File.AddNewItem	CTRL + MAJ + A	Affiche la boîte de dialogue **Ajouter un nouvel élément**, dans laquelle vous pouvez ajouter un nouveau fichier au projet en cours.
File.Exit	ALT + Q	Quitte l'environnement de développement intégré.
File.Print	CTRL + P	Affiche la boîte de dialogue **Imprimer**, dans laquelle vous pouvez sélectionner des paramètres d'impression.
File.SaveAll	CTRL + MAJ + S	Enregistre tous les documents de la solution en cours et les fichiers éventuels dans le projet des fichiers externes.
File.SaveSelectedItems	CTRL + S	Enregistre les éléments sélectionnés dans le projet en cours.
Tools.GoToCommandLine	CTRL + /	Place le signe insertion dans la zone **Rechercher/Commande** de la barre d'outils standard.
View.NavigateBackward	CTRL + MAJ + F2	Retourne au document précédent ou à la fenêtre précédente dans l'historique de navigation.
View.NavigateForward	CTRL + MAJ + -	Passe au document suivant ou à la fenêtre suivante dans l'historique de navigation.
View.WebNavigateBack	ALT + GAUCHE	Affiche la page précédente dans l'historique d'affichage. Disponible seulement dans le navigateur Web intégré.
View.WebNavigateForward	ALT + DROITE	Affiche la page suivante de l'historique d'affichage. Disponible seulement dans le navigateur Web intégré.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de projet, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de l'aide intégrée, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de l'aide intégrée, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes pour afficher les rubriques d'aide et vous déplacer entre les rubriques.

Nom de la commande	Touches de raccourci	Description
Help.Contents	CTRL + ALT + F1	Affiche la fenêtre **Sommaire** de la documentation contenue dans MSDN.
Help.DynamicHelp	CTRL + F1	Affiche la fenêtre **Aide dynamique**, qui affiche des rubriques différentes en fonction des éléments ayant actuellement le focus dans le produit.
Help.F1Help	F1	Affiche une rubrique d'aide relative à l'interface utilisateur sélectionnée.
Help.Index	CTRL + ALT + F2	Affiche la fenêtre **Index** pour la documentation contenue dans MSDN.
Help.Nexttopic	ALT + BAS	Affiche la rubrique suivante dans le sommaire. Disponible uniquement dans la fenêtre **Aide** du navigateur Web.
Help.Previoustopic	ALT + HAUT	Affiche la rubrique précédente dans le sommaire. Disponible uniquement dans la fenêtre **Aide** du navigateur Web.
Help.Search	CTRL + ALT + F3	Affiche la fenêtre **Rechercher**, qui vous permet de rechercher des mots ou des expressions dans la documentation contenue dans MSDN.
Help.WindowHelp	MAJ + F1	Affiche une rubrique d'aide relative à l'interface utilisateur sélectionnée.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de la gestion des fenêtres, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de projet, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes pour ajouter de nouveaux éléments à un projet, créer un nouveau projet ou ouvrir des fichiers ou des projets.

Nom de la commande	Touches de raccourci	Description
Build.BuildSolution	CTRL + MAJ + B	Génère tous les projets dans la solution avec la configuration de solution en cours.
Build.Compile	CTRL + F7	Crée un fichier objet contenant le code machine, les directives de l'éditeur de liens, des sections, des références externes et les noms des fonctions/données pour le fichier sélectionné.
File.NewFile	CTRL + MAJ + N	Affiche la boîte de dialogue **Nouveau fichier**, dans laquelle vous pouvez sélectionner un nouveau fichier à ajouter au projet en cours.
File.NewProject	CTRL + N	Affiche la boîte de dialogue **Nouveau projet**, dans laquelle vous pouvez créer des projets et les ajouter à la solution en cours.
File.OpenFile	CTRL + MAJ + O	Affiche la boîte de dialogue **Ouvrir un fichier**, dans laquelle vous pouvez sélectionner un fichier existant en vue de l'ouvrir.
File.OpenProject	CTRL + O	Affiche la boîte de dialogue **Ouvrir un projet**, dans laquelle vous pouvez ajouter des projets existants à votre solution.
File.AddExistingItem	CTRL + D	Affiche la boîte de dialogue **Ajouter un élément existant** qui permet d'ajouter un fichier existant au projet en cours.
File.AddNewItem	CTRL + MAJ + A	Affiche la boîte de dialogue **Ajouter un nouvel élément**, dans laquelle vous pouvez ajouter un nouveau fichier au projet en cours.
Project.RunSelection	CTRL + Q	Exécute la sélection en cours.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci globales, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de l'aide intégrée, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de la gestion des fenêtres, option contextuelle par défaut de Visual Basic 6.0

Concepts

Introduction aux solutions, projets et éléments

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci du Concepteur HTML, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes uniquement pendant une opération d'édition dans le Concepteur HTML. Certaines combinaisons ne sont disponibles que dans un mode spécifique de ce concepteur.

Commande	**Touches de raccourci**	**Description**
Edit.CharTranspose	CTRL + T	Permute les caractères situés de part et d'autre du point d'insertion. Par exemple, AC|BD devient AB|CD. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Edit.ClearAllBookmarks	CTRL + K, CTRL + L	Supprime tous les signets sans nom dans le document actif. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Edit.CompleteWord	CTRL + ESPACE	Exécute le mot dans le langage actuel. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Edit.MakeLowercase	CTRL + U	Convertit en minuscules les caractères du texte sélectionné. Disponible uniquement en **mode HTML** dans le **Concepteur HTML**.
Edit.MakeUppercase	CTRL + MAJ + U	Convertit en majuscules les caractères du texte sélectionné. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Edit.NextBookmark	CTRL + K, CTRL + N	Déplace le curseur vers le signet suivant dans le document. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Edit.WordDeleteToEnd	CTRL + SUPPR CTRL + MAJ + RET. ARR	Supprime le mot situé à droite du point d'insertion. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Edit.WordDeleteToStart	CTRL + RET. ARR	Supprime le mot situé à gauche du point d'insertion. Disponible uniquement en mode **HTML** dans le **Concepteur HTML**.
Format.Bold	CTRL + B	Active ou désactive le gras pour le texte sélectionné. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.DecreaseIndent	CTRL + MAJ + T	Réduit d'une unité le retrait du paragraphe sélectionné. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.

Format.IncreaseIndent	CTRL + T	Met le paragraphe sélectionné en retrait d'un niveau. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.Italic	CTRL + I	Active ou désactive l'italique pour le mot sélectionné. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.LockElement	CTRL + MAJ + K	Empêche un élément ayant une position absolue d'être déplacé par inadvertance. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.ShowGrid	CTRL + G	Active ou désactive le mode Aligner sur la grille. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.SnaptoGrid	CTRL + MAJ + G	Spécifie que les éléments sont alignés par rapport à une grille invisible. Vous pouvez définir l'espacement de la grille en sélectionnant depuis la boîte de dialogue Options le volet **Afficher** dans les options du Concepteur HTML. La grille sera modifiée lors de la prochaine ouverture d'un document. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.Underline	CTRL + U	Active ou désactive le souligné pour le texte sélectionné. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Format.InsertBookmark	CTRL + MAJ + L	Affiche la boîte de dialogue Signet et permet de créer une cible pour les renvois au sein du document en cours. Disponible en mode **Design** et en mode **HTML** du **Concepteur HTML**.
Format.ConverttoHyperlink	CTRL + L	Quand le texte est sélectionné, affiche la boîte de dialogue Lien hypertexte. Disponible en mode **Design** et en mode **HTML** du **Concepteur HTML**.
Table.InsertRowAbove	CTRL + ALT + HAUT	Ajoute une ligne au-dessus de la ligne en cours dans la table. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Table.InsertRowBelow	CTRL + ALT + BAS	Ajoute une ligne au-dessous de la ligne en cours dans la table. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Table.InsertColumnstotheLeft	CTRL + ALT + GAUCHE	Ajoute une colonne à gauche de la colonne en cours dans la table. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
Table.InsertColumnstotheRight	CTRL + ALT + DROITE	Ajoute une colonne à droite de la colonne en cours dans la table. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
View.Details	CTRL + MAJ + Q	Affiche des icônes signalant l'emplacement des éléments HTML qui ne s'affichent pas, tels que les commentaires, les scripts et les ancres des éléments ayant une position absolue. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.

View.NextView	CTRL + PG.SUIV	Passe du mode **Design** au mode **HTML** et inversement. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.
View.VisibleBorders	CTRL + Q	Affiche une bordure de 1 pixel autour des éléments HTML (par exemple, les tableaux, les cellules de tableau et les divisions) pour lesquels un attribut BORDER a la valeur zéro. Disponible uniquement en mode **Design** dans le **Concepteur HTML**.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Concepteur HTML

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de l'Éditeur d'images, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes dans l'Éditeur d'images.

Commande	Touches de raccourci	Description
Image.DrawOpaque	CTRL + J	Rend la sélection en cours opaque ou transparente. Disponible uniquement dans l'**Éditeur d'images**.
Image.FlipHorizontal	CTRL + H	Retourne l'image de droite à gauche sur l'axe horizontal. Disponible uniquement dans l'**Éditeur d'images**.
Image.FlipVertical	MAJ + ALT + H	Retourne l'image de haut en bas sur l'axe vertical. Disponible uniquement dans l'**Éditeur d'images**.
Image.NewImageType	INSER	Affiche la boîte de dialogue **Nouveau type d'image icône**, qui vous permet de sélectionner un nouveau type d'image à créer. Disponible uniquement dans l'**Éditeur d'images**.
Image.Rotate90Degrees	CTRL + MAJ + H	Fait pivoter une image de 90 degrés dans le sens des aiguilles d'une montre. Disponible uniquement dans l'**Éditeur d'images**.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Éditeur d'images

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de débogage, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes pendant le débogage du code.

Commande	**Touches de raccourci**	**Description**
Debug.ApplyCodeChanges	ALT + F10	Applique les modifications apportées au code sans arrêter le débogueur.
Debug.Autos	CTRL + ALT + V, A	Affiche la fenêtre **Automatique** qui présente les valeurs des variables de la ligne en cours d'exécution dans la procédure active.
Debug.BreakAll	CTRL + ARRÊT	Arrête temporairement l'exécution de tous les processus au cours d'une session de débogage. Disponible uniquement en mode exécution.
Debug.Breakpoints	CTRL + ALT + B	Affiche la boîte de dialogue **Points d'arrêt**, dans laquelle vous pouvez ajouter et modifier des points d'arrêt.
Debug.CallStack	CTRL + L	Affiche la fenêtre **Pile des appels** qui présente la liste de toutes les procédures ou de tous les frames de pile actifs pour le thread d'exécution en cours. Disponible uniquement en mode exécution.
Debug.ClearAllBreakpoints	CTRL + MAJ + F9	Efface tous les points d'arrêt du projet.
Debug.Disassembly	CTRL + ALT + D	Affiche la boîte de dialogue **Désassemblage**.
Debug.Exceptions	CTRL + ALT + E	Affiche la boîte de dialogue **Exceptions**.
Debug.Immediate	CTRL + G	Affiche la fenêtre **Exécution** dans laquelle vous pouvez évaluer des expressions et exécuter des commandes individuelles.
Debug.Locals	CTRL + ALT + V, L	Affiche la fenêtre **Variables locales**, qui présente les variables et leurs valeurs pour chaque procédure du frame de pile en cours.
Debug.Memory1	CTRL + ALT + M, 1	Affiche la fenêtre **Mémoire 1**, qui permet de visualiser les tampons de grande taille, les chaînes et toutes les autres données qui ne s'affichent pas correctement dans la fenêtre **Espion** ou **Variables**.
Debug.Memory2	CTRL + ALT + M, 2	Affiche la fenêtre **Mémoire 2**, qui permet de visualiser les tampons de grande taille, les chaînes et toutes les autres données qui ne s'affichent pas correctement dans la fenêtre **Espion** ou **Variables**.

Debug.Memory3	CTRL + ALT + M, 3	Affiche la fenêtre **Mémoire 3**, qui permet de visualiser les tampons de grande taille, les chaînes et toutes les autres données qui ne s'affichent pas correctement dans la fenêtre **Espion** ou **Variables**.
Debug.Memory4	CTRL + ALT + M, 4	Affiche la fenêtre Mémoire 4, qui permet de visualiser les tampons de grande taille, les chaînes et toutes les autres données qui ne s'affichent pas correctement dans la fenêtre **Espion** ou **Variables**.
Debug.Modules	CTRL + ALT + U	Affiche la fenêtre **Modules**, qui vous permet d'afficher les fichiers .dll ou .exe utilisés par le programme.
Debug.NewBreakpoint	CTRL + B	Insère ou efface un point d'arrêt sur la ligne de code active.
Debug.QuickWatch	MAJ + F9	Affiche la boîte de dialogue **Espion express** qui présente la valeur actuelle de l'expression sélectionnée. Disponible uniquement en mode Arrêt. Utilisez cette commande pour vérifier la valeur actuelle d'une variable, d'une propriété ou de toute autre expression pour laquelle vous n'avez défini aucune expression espionne.
Debug.Registers	CTRL + ALT + G	Affiche la fenêtre **Registres** qui présente le contenu pour le débogage des applications en code natif.
Debug.Restart	MAJ + F5	Arrête une session de débogage, régénère le code, puis exécute à nouveau l'application depuis le début. Disponible en mode arrêt et en mode exécution.
Debug.RunningDocuments	CTRL + ALT + N	Affiche la fenêtre **Documents en cours d'exécution** qui présente l'ensemble de documents en cours de débogage. Disponible en mode exécution.
Debug.RunToCursor	CTRL + F8	En mode Arrêt, reprend l'exécution de votre code à partir de l'instruction en cours, jusqu'à l'instruction sélectionnée. L'indicateur en marge **Ligne actuellement exécutée** apparaît dans la **Barre des indicateurs en marge**. En mode Design, démarre le débogueur et exécute votre code jusqu'à l'emplacement du curseur.
Debug.SetNextStatement	CTRL + F9	Définit le point d'exécution au niveau de la ligne de code choisie.
Debug.ShowNextStatement	ALT + NUM *	Met en surbrillance l'instruction suivante à exécuter.
Debug.Start	F5	Attache automatiquement le débogueur et exécute l'application à partir du formulaire de démarrage spécifié dans la boîte de dialogue **Propriétés de <Projet>**. En mode Arrêt, permet de continuer.
Debug.StartWithoutDebugging	CTRL + F5	Exécute le code sans faire appel au débogueur.
Debug.StepInto	F8	Exécute le code instruction par instruction en suivant l'exécution d'appels de fonction.
Debug.StepOut	CTRL + MAJ + F8	Exécute les lignes restantes d'une fonction dans laquelle se trouve le point d'exécution en cours.

Debug.StepOver	MAJ + F8	Exécute la ligne de code suivante, sans suivre l'exécution d'appels de fonction.
Debug.This	CTRL + ALT + V, T	Affiche la fenêtre **This** qui vous permet de visualiser les données membres de l'objet associé à la méthode en cours.
Debug.Threads	CTRL + ALT + H	Affiche la fenêtre **Threads** qui répertorie tous les threads du processus en cours et fournit des informations à leur sujet.
Debug.ToggleBreakpoint	F9	Définit ou supprime un point d'arrêt sur la ligne active.
Debug.ToggleDisassembly	CTRL + F11	Affiche les informations de code machine pour le fichier source en cours. Disponible uniquement en mode Arrêt.
Tools.DebugProcesses	CTRL + ALT + P	Affiche la boîte de dialogue **Processus** qui vous permet de déboguer plusieurs programmes au même moment dans une solution unique.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de la fenêtre Outil, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de recherche et remplacement, option contextuelle par défaut de Visual Basic 6.0

Les combinaisons de touches de raccourci suivantes peuvent être utilisées avec les boîtes de dialogue **Rechercher**, **Remplacer**, **Rechercher dans les fichiers** et **Remplacer dans les fichiers**.

Commande	**Touches de raccourci**	**Description**
Edit.Find	CTRL + F	Affiche la boîte de dialogue **Rechercher**.
Edit.FindNext	F3	Recherche l'occurrence suivante du texte de la recherche précédente.
Edit.FindNextSelected	CTRL + F3	Recherche l'occurrence suivante du texte sélectionné dans le document.
Edit.FindPrevious	MAJ + F3	Recherche l'occurrence précédente du texte recherché.
Edit.FindPreviousSelected	CTRL + MAJ + F3	Recherche soit l'occurrence précédente du texte actuellement sélectionné, soit le mot à l'emplacement du signe insertion.
Edit.GoToFindCombo	CTRL + MAJ + F	Place le signe insertion sur la ligne **Rechercher/Commande** de la barre d'outils standard.
Edit.HiddenText	ALT + F3, H	Active ou désactive l'option **Rechercher le texte masqué** pour la boîte de dialogue **Rechercher**.
Edit.IncrementalSearch	CTRL + ALT + I	Commence une recherche incrémentielle. Si une recherche incrémentielle est lancée, mais si vous n'avez tapé aucun caractère, rappelle le modèle de recherche précédent. Si le texte a été trouvé, recherche l'occurrence suivante.
Edit.MatchCase	ALT + F3, + C	Active ou désactive l'option **Respecter la casse** pour les opérations **Rechercher** et **Remplacer**.
Edit.RegularExpression	ALT + F3, + R	Ative ou désactive l'option **Expression régulière** afin que les caractères spéciaux puissent être utilisés dans les opérations **Rechercher** et **Remplacer**.
Edit.Replace	CTRL + H	Affiche la boîte de dialogue **Remplacer**.
Edit.ReplaceinFiles	CTRL + MAJ + H	Affiche la boîte de dialogue **Remplacer dans les fichiers**.
Edit.ReverseIncrementalSearch	CTRL + MAJ + ALT + I	Change le sens de la recherche incrémentielle pour qu'elle s'effectue de bas en haut dans le fichier.
Edit.StopSearch	ALT + F3, + S	Arrête l'opération **Rechercher dans les fichiers** en cours.
Edit.Up	ALT + F3, + B	Active ou désactive l'option **Rechercher vers le haut** pour les opérations **Rechercher** et **Remplacer**.

Edit.WholeWord	ALT + F3, + W	Active ou désactive l'option **Mot entier** pour les opérations **Rechercher** et **Remplacer**.
Edit.Wildcard	ALT + F3, P	Active ou désactive l'option **Caractère générique** pour les opérations **Rechercher** et **Remplacer**.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de navigation dans un texte, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de sélection de texte, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de manipulation de texte, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de Visual Database Tools, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes dans le Concepteur de base de données ou le Concepteur de requêtes.

Commande	Touches de raccourci	Description
Database.Run	CTRL + R	Exécute l'objet de base de données actuellement sélectionné.
Database.StepInto	CTRL + E	Passe en mode débogage pour l'objet de base de données actif.
Query.Run	CTRL + R	Exécute la requête. Disponible uniquement dans Visual Database Tools.
View.Diagram	CTRL + 1	Affiche le volet **Schéma** du Concepteur de requêtes. Disponible seulement dans le Concepteur de requêtes et dans le Concepteur de vues.
View.Grid	CTRL + 2	Affiche le volet **Grille** du Concepteur de requêtes. Disponible seulement dans le Concepteur de requêtes et dans le Concepteur de vues.
View.Results	CTRL + 4	Affiche le volet **Résultats** du Concepteur de requêtes. Disponible seulement dans le Concepteur de requêtes et dans le Concepteur de vues.
View.SQL	CTRL + 3	Affiche le volet **SQL** du Concepteur de requêtes. Disponible seulement dans le Concepteur de requêtes et dans le Concepteur de vues.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci globales, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de l'Explorateur d'objets, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes dans la fenêtre Explorateur d'objets.

Commande	**Touches de raccourci**	**Description**
Edit.FindSymbol	CTRL + MAJ+ Y CTRL + MAJ + ALT + F12	Affiche la boîte de dialogue **Rechercher un symbole**.
View.FindSymbolResults	CTRL + ALT + Y	Affiche la fenêtre **Résultats de la recherche de symbole**.
View.ObjectBrowserBack	ALT + -	Revient à l'objet sélectionné précédemment dans l'historique de sélection de l'**Explorateur d'objets**.
View.ObjectBrowserForward	MAJ + ALT + -	Passe à l'objet suivant dans l'historique de sélection de l'**Explorateur d'objets**.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de la fenêtre Outil, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de la gestion des fenêtres, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de manipulation des contrôles, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes pour déplacer les contrôles, les sélectionner et changer leur taille sur les aires de conception.

Commande	**Touches de raccourci**	**Description**
Edit.MoveControlDown	CTRL + BAS	Descend le contrôle sélectionné par incréments de 1 dans l'aire de conception.
Edit.MoveControlDownGrid	BAS	Descend le contrôle sélectionné par incréments de 8 dans l'aire de conception.
Edit.MoveControlLeft	CTRL + GAUCHE	Déplace le contrôle vers la gauche par incréments de 1 sur l'aire de conception.
Edit.MoveControlLeftGrid	GAUCHE	Déplace le contrôle vers la gauche par incréments de 8 sur l'aire de conception.
Edit.MoveControlRight	CTRL + DROITE	Déplace le contrôle sélectionné vers la droite par incréments de 1 dans l'aire de conception.
Edit.MoveControlRightGrid	DROITE	Déplace le contrôle vers la droite par incréments de 8 sur l'aire de conception.
Edit.MoveControlUp	CTRL + HAUT	Remonte le contrôle sélectionné par incréments de 1 dans l'aire de conception.
Edit.MoveControlUpGrid	HAUT	Remonte le contrôle sélectionné par incréments de 8 dans l'aire de conception.
Edit.SelectNextControl	TAB	Sélectionne le contrôle suivant sur l'aire de conception.
Edit.SelectPreviousControl	MAJ + TAB	Sélectionne le contrôle précédemment sélectionné sur l'aire de conception.
Edit.SizeControlDown	CTRL + MAJ + BAS	Augmente la hauteur du contrôle par incréments de 1 dans l'aire de conception.
Edit.SizeControlDownGrid	MAJ + BAS	Augmente la hauteur du contrôle par incréments de 8 dans l'aire de conception.
Edit.SizeControlLeft	CTRL + MAJ + GAUCHE	Réduit la largeur du contrôle par incréments de 1 dans l'aire de conception.
Edit.SizeControlLeftGrid	MAJ + GAUCHE	Réduit la largeur du contrôle par incréments de 8 dans l'aire de conception.
Edit.SizeControlRight	CTRL + MAJ + DROITE	Augmente la largeur du contrôle par incréments de 1 dans l'aire de conception.
Edit.SizeControlRightGrid	MAJ + DROITE	Augmente la largeur du contrôle par incréments de 8 dans l'aire de conception.
Edit.SizeControlUp	CTRL + MAJ + HAUT	Diminue la hauteur du contrôle par incréments de 1 dans l'aire de conception.
Edit.SizeControlUpGrid	MAJ + HAUT	Diminue la hauteur du contrôle par incréments de 8 dans l'aire de conception.

Format.BringToFront	CTRL + J	Déplace le contrôle sélectionné en haut de l'ordre z.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci globales, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de navigation dans un texte, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes dans les éditeurs de texte pour vous déplacer à l'intérieur d'un document ouvert.

Commande	Touches de raccourci	Description
Edit.CharLeft	GAUCHE	Déplace le curseur d'un caractère vers la gauche.
Edit.CharRight	DROITE	Déplace le curseur d'un caractère vers la droite.
Edit.DocumentEnd	CTRL + FIN	Place le point d'insertion sur la dernière ligne du document.
Edit.DocumentStart	CTRL + DÉBUT	Place le point d'insertion sur la première ligne du document.
Edit.GotoBrace	CTRL +]	Place le point d'insertion sur l'accolade suivante dans le document.
Edit.GoToDeclaration	CTRL + ALT + F12	Affiche la définition du symbole sélectionné dans le code.
Edit.GoToDefinition	MAJ + F2	Affiche la déclaration du symbole sélectionné dans le code.
Edit.LineDown	BAS	Place le point d'insertion sur la ligne suivante.
Edit.LineEnd	FIN	Place le point d'insertion à la fin de la ligne.
Edit.LineStart	DÉBUT	Place le point d'insertion au début de la ligne.
Edit.LineUp	HAUT	Place le point d'insertion sur la ligne précédente.
Edit.PageDown	PG.SUIV	Faite défiler le document d'une page vers le bas.
Edit.PageUp	PG.PRÉC	Faite défiler le document d'une page vers le haut.
Edit.PreviousBookmark	CTRL + K, CTRL + P	Déplace le curseur vers le signet précédent.
Edit.ScrollLineDown	CTRL + BAS	Fait défiler le texte d'une ligne vers le bas.
Edit.ScrollLineUp	CTRL + HAUT	Fait défiler le texte d'une ligne vers le haut.
Edit.ToggleWordWrap	CTRL + R, CTRL + R	Active ou désactive le retour automatique à la ligne dans un éditeur.
Edit.ViewBottom	CTRL + PG.SUIV	Déplace le curseur en bas du document actif.
Edit.ViewTop	CTRL + PG.PRÉC	Se place en haut de la fenêtre active ou du document actif.
Edit.WordNext	CTRL + DROITE	Déplace le point d'insertion d'un mot vers la droite.
Edit.WordPrevious	CTRL + GAUCHE	Déplace le point d'insertion d'un mot vers la gauche.
View.ViewCode	F7	Affiche l'élément sélectionné dans l'Éditeur en mode Code.

View.ViewDesigner	MAJ + F7	Affiche l'élément sélectionné dans l'éditeur en mode Design.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de sélection de texte, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de manipulation de texte, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de sélection de texte, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes dans les éditeurs de texte pour sélectionner du texte à l'intérieur d'un document ouvert.

Commande	**Touches de raccourci**	**Description**
Edit.CharLeftExtend	MAJ + GAUCHE	Étend la sélection en déplaçant le curseur d'un caractère vers la gauche.
Edit.CharLeftExtendColumn	MAJ + ALT + GAUCHE	Étend la sélection de la colonne en déplaçant le curseur d'un caractère vers la gauche.
Edit.CharRightExtend	MAJ + DROITE	Étend la sélection en déplaçant le curseur d'un caractère vers la droite.
Edit.CharRightExtendColumn	MAJ + ALT + DROITE	Étend la sélection de la colonne en déplaçant le curseur d'un caractère vers la droite.
Edit.DocumentEndExtend	CTRL + MAJ + FIN	Sélectionne le texte du point d'insertion jusqu'à la dernière ligne du document.
Edit.DocumentStartExtend	CTRL + MAJ + DÉBUT	Sélectionne le texte du point d'insertion jusqu'à la première ligne du document.
Edit.GotoBraceExtend	CTRL + MAJ +]	Étend la sélection en plaçant le point d'insertion sur l'accolade suivante.
Edit.LineDownExtend	MAJ + BAS	Étend la sélection du texte d'une ligne vers le bas, à partir de l'emplacement du point d'insertion.
Edit.LineDownExtendColumn	MAJ + ALT + BAS	Étend la sélection de la colonne en déplaçant le curseur d'une ligne vers le bas.
Edit.LineEndExtend	MAJ + FIN	Sélectionne le texte du point d'insertion jusqu'à la fin de la ligne active.
Edit.LineEndExtendColumn	MAJ + ALT + FIN	Étend la sélection de la colonne en plaçant le point d'insertion à la fin de la ligne.
Edit.LineStartExtend	MAJ + DÉBUT	Sélectionne le texte du point d'insertion jusqu'au début de la ligne.
Edit.LineUpExtend	MAJ + HAUT	Sélectionne le texte ligne par ligne vers le haut, en partant de l'emplacement du point d'insertion.
Edit.LineUpExtendColumn	MAJ + ALT + HAUT	Étend la sélection de la colonne en déplaçant le curseur d'une ligne vers le haut.
Edit.PageDownExtend	MAJ + PG.SUIV	Étend la sélection d'une page vers le bas.
Edit.PageUpExtend	MAJ + PG.SUIV	Étend la sélection d'une page vers le haut.
Edit.SelectAll	CTRL + A	Sélectionne la totalité du document actif.
Edit.SelectCurrentWord	CTRL + W	Sélectionne le mot contenant le point d'insertion ou le mot situé à droite du point d'insertion.

Edit.SelectToLastGoBack	CTRL + =	Sélectionne tout ce qui est compris entre l'emplacement actuel et l'emplacement précédent dans l'éditeur.
Edit.ViewBottomExtend	CTRL + MAJ + PG.SUIV	Étend la sélection en plaçant le curseur sur la dernière ligne affichée.
Edit.ViewTopExtend	CTRL + MAJ + PG.PRÉC	Étend la sélection jusqu'en haut de la fenêtre active ou du document actif.
Edit.WordNextExtend	CTRL + MAJ + DROITE	Étend la sélection d'un mot vers la droite.
Edit.WordNextExtendColumn	CTRL + MAJ + ALT + DROITE	Étend la sélection de la colonne en déplaçant le curseur d'un mot vers la droite.
Edit.WordPreviousExtend	CTRL + MAJ + GAUCHE	Étend la sélection d'un mot vers la gauche.
Edit.WordPreviousExtendColumn	CTRL + MAJ + ALT + GAUCHE	Étend la sélection de la colonne en déplaçant le curseur d'un mot vers la gauche.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de navigation dans un texte, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de manipulation de texte, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de manipulation de texte, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes dans les éditeurs de texte pour supprimer, déplacer ou mettre en forme du texte à l'intérieur d'un document ouvert.

Commande	**Touches de raccourci**	**Description**
Edit.BreakLine	ENTRÉE MAJ + ENTRÉE	Insère une nouvelle ligne.
Edit.CollapsetoDefinitions	CTRL + M, CTRL + O	Détermine automatiquement les limites logiques pour créer des régions dans le code (par exemple, des procédures), puis les masque.
Edit.CommentSelection	CTRL + K, CTRL + C	Marque la ligne de code active comme un commentaire, à l'aide de la syntaxe de commentaire correcte pour le langage de programmation.
Edit.DeleteHorizontalWhiteSpace	CTRL + K, CTRL + \	Réduit l'espace blanc dans la sélection ou supprime l'espace blanc contigu au curseur s'il n'y a aucune sélection.
Edit.FormatDocument	CTRL + K, CTRL + D	Applique la mise en retrait et la mise en forme des espaces pour le langage, comme spécifié dans le volet Mise en forme de ce langage à l'intérieur de la section Éditeur de texte de la boîte de dialogue Options.
Edit.FormatSelection	CTRL + K, CTRL + F	Met correctement en retrait les lignes de code sélectionnées par rapport aux lignes de code situées autour.
Edit.HideSelection	CTRL + M, CTRL + H	Masque le texte sélectionné. Une icône signale l'emplacement du texte masqué dans le fichier.
Edit.InsertTab	TAB	Met la ligne de texte en retrait du nombre spécifié d'espaces, par exemple de cinq espaces. Disponible seulement dans le Concepteur .NET Framework.
Edit.LineCut	CTRL + Y	Coupe toutes les lignes sélectionnées, ou la ligne active si aucune ligne n'a été sélectionnée, et les place dans le Presse-papiers.
Edit.LineDelete	CTRL + MAJ+ Y	Supprime toutes les lignes sélectionnées ou la ligne active si aucune ligne n'a été sélectionnée.
Edit.LineOpenAbove	CTRL + ENTRÉE	Insère une ligne vide au-dessus du point d'insertion.
Edit.LineOpenBelow	CTRL + MAJ + ENTRÉE	Insère une ligne vide en dessous du point d'insertion.
Edit.LineTranspose	MAJ + ALT + T	Place la ligne contenant le point d'insertion en dessous de la ligne suivante.
Edit.OvertypeMode	INSER	Tape les caractères par-dessus les caractères déjà présents dans le document, au lieu de les insérer. Disponible uniquement dans les éditeurs de texte.

Edit.StopHidingCurrent	CTRL + M, CTRL + U	Supprime les informations en mode Plan pour la zone sélectionnée.
Edit.StopOutlining	CTRL + M, CTRL + P	Supprime toutes les informations en mode Plan pour le document complet.
Edit.SwapAnchor	CTRL + R, CTRL + P	Permute l'ancre et les points de fin de la sélection en cours.
Edit.TabLeft	MAJ + TAB	Déplace les lignes sélectionnées d'un taquet de tabulation vers la gauche. Disponible seulement dans le Concepteur .NET Framework.
Edit.ToggleAllOutlining	CTRL + M, CTRL + L	Active ou désactive toutes les sections de texte précédemment marquées comme masquées et affiche les états.
Edit.ToggleBookmark	CTRL + K, CTRL + K	Définit ou supprime un signet sur la ligne active.
Edit.ToggleOutliningExpansion	CTRL + M, CTRL + M	Active ou désactive la section de texte masquée actuellement sélectionnée.
Edit.ToggleTaskListShortcut	CTRL + K, CTRL + H	Définit ou supprime un raccourci dans la ligne active.
Edit.UncommentSelection	CTRL + K, CTRL + U	Supprime la syntaxe du commentaire de la ligne de code active.
Edit.ViewWhiteSpace	CTRL + R, CTRL + W	Affiche ou masque les espaces et les marques de tabulation.
Edit.WordTranspose	CTRL + MAJ + T	Permute les mots situés de part et d'autre du point d'insertion.
View.ShowReferences	ALT + F12	Effectue une recherche de symbole avec les options Respecter la casse et Mot entier, puis affiche les résultats dans la fenêtre Résultats de la recherche de symbole.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de navigation dans un texte, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de sélection de texte, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de la gestion des fenêtres, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes pour déplacer ou fermer les fenêtres Outil et les fenêtres de document, ou encore pour naviguer à l'intérieur de ces fenêtres.

Commande	Touches de raccourci	Description
View.FullScreen	MAJ + ALT + ENTRÉE	Active ou désactive le mode plein écran.
Window.ActivateDocumentWindow	ÉCHAP	Ferme un menu ou une boîte de dialogue, annule une opération en cours ou met le focus dans la fenêtre de document active.
Window.CloseDocumentWindow	CTRL + F4	Ferme la fenêtre enfant MDI active.
Window.CloseToolWindow	MAJ + ÉCHAP	Ferme la fenêtre Outil active.
Window.MoveToDropDownBar	CTRL + F2	Place le curseur sur la barre déroulante lorsque l'éditeur est en mode Code.
Window.NextDocumentWindow	CTRL + F6 CTRL + TAB	Fait défiler les fenêtres enfants MDI une par une.
Window.NextPane	CTRL + F12	Place le curseur dans la fenêtre Outil suivante.
Window.NextSplitPane	F6	Se place dans le volet suivant d'un document comportant plusieurs volets de fractionnement.
Window.NextTab	CTRL + PG.SUIV	Déplace le curseur dans l'onglet suivant du document ou de la fenêtre.
Window.PreviousDocumentWindow	CTRL + MAJ+ F6 CTRL + MAJ+ TAB	Effectue un déplacement vers le document précédent à l'intérieur de l'éditeur ou du concepteur.
Window.PreviousPane	CTRL + MAJ + F12	Place le curseur dans la fenêtre précédemment sélectionnée.
Window.PreviousSplitPane	MAJ + F6	Déplace le curseur dans le volet précédent d'un document comportant plusieurs volets de fractionnement.
Window.PreviousTab	CTRL + PG.PRÉC	Fait passer le curseur dans l'onglet précédent du document ou de la fenêtre.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de la fenêtre Outil, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci globales, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de macro, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes avec des macros.

Commande	Touches de raccourci	Description
Tools.MacrosIDE	ALT + F11	Démarre l'éditeur de macros, **Microsoft Visual Studio Macros**.
Tools.RecordTemporaryMacro	CTRL + MAJ + R	Place l'environnement en mode d'enregistrement de macro.
Tools.RunTemporaryMacro	CTRL + MAJ + P	Réexécute une macro enregistrée.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Référence

Touches de raccourci de manipulation de texte, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de la gestion des fenêtres, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de la fenêtre Outil, option contextuelle par défaut de Visual Basic 6.0

Touches de raccourci de projet, option contextuelle par défaut de Visual Basic 6.0

Autres ressources

Visual Studio Macros

Touches de raccourci par défaut de Visual Basic 6.0

Touches de raccourci de la fenêtre Outil, option contextuelle par défaut de Visual Basic 6.0

Vous pouvez utiliser les combinaisons de touches de raccourci suivantes pour afficher des fenêtres Outil spécifiques.

Commande	Touches de raccourci	Description
Tools.CommandWindowMarkMode	CTRL + MAJ + M	Place la fenêtre **Commande** dans un mode qui autorise la sélection de texte à l'intérieur de la fenêtre.
Tools.AddRemoveToolboxItems	CTRL + T	Affiche la boîte de dialogue **Personnaliser la boîte à outils** qui vous permet d'ajouter et de supprimer des éléments dans la Boîte à outils.
View.MacroExplorer	ALT + F8	Affiche la fenêtre **Explorateur de macros**, qui présente toutes les macros disponibles dans la solution en cours.
View.ClassView	CTRL + MAJ + C	Affiche la fenêtre **Affichage de classes**.
View.CommandWindow	CTRL + ALT + A	Affiche la fenêtre **Commande**, dans laquelle vous pouvez taper des commandes qui manipulent l'environnement de développement intégré (IDE).
View.DocumentOutline	CTRL + ALT + T	Affiche la fenêtre **Structure du document** qui présente la structure plate ou hiérarchique du document actif.
View.Favorites	CTRL + ALT + F	Affiche la fenêtre **Favoris** qui répertorie tous les raccourcis vers les pages Web.
View.ObjectBrowser	F2	Affiche l'**Explorateur d'objets**, qui présente les classes, propriétés, méthodes, événements et constantes disponibles pour les packages, ainsi que les bibliothèques d'objets et les procédures de votre projet.
View.Output	CTRL + ALT + O	Affiche la fenêtre **Sortie**, dans laquelle vous pouvez voir les messages d'état au moment de l'exécution.
View.PropertiesWindow	F4	Affiche la fenêtre **Propriétés**, qui répertorie les propriétés et les événements au moment du design pour l'élément actuellement sélectionné.
View.PropertyPages	MAJ + F4	Affiche les pages de propriétés pour l'élément actuellement sélectionné.
View.ResourceView	CTRL + MAJ + E	Affiche la fenêtre **Affichage des ressources**.
View.ServerExplorer	CTRL + ALT + S	Affiche l'**Explorateur de serveurs**, qui vous permet d'afficher et de manipuler des serveurs de base de données, des journaux d'événements, des files d'attente de messages, des services Web et de nombreux autres services du système d'exploitation.
View.SolutionExplorer	CTRL + R	Affiche l'**Explorateur de solutions** qui dresse la liste des projets et des fichiers de la solution en cours.

View.TaskList	CTRL + ALT + K	Affiche la fenêtre **Liste des tâches** où vous pouvez personnaliser, classer et gérer des tâches, des commentaires, des raccourcis, des avertissements et des messages d'erreur.
View.Toolbox	CTRL + ALT + X	Affiche la **boîte à outils**, qui contient des contrôles et d'autres éléments susceptibles d'être inclus ou utilisés dans votre code.

Voir aussi

Tâches

Comment : utiliser les combinaisons de touches de raccourci

Autres ressources

Touches de raccourci par défaut de Visual Basic 6.0

Aide destinée aux gestionnaires d'événements dans le code Visual Basic

Pour obtenir de l'aide sur un gestionnaire d'événements particulier tandis que vous vous trouvez dans l'éditeur de code, placez votre curseur sur la clause **Handles** à l'extrémité arrière de la procédure événementielle, puis appuyez sur F1. Par exemple, dans l'instruction `Form1_Load` ci-dessous, l'emplacement où vous devez placer le curseur est le terme en gras qui suit `Handles` :

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles MyBase.Load

End Sub
```

Voir aussi

Concepts

Vue d'ensemble des gestionnaires d'événements (Windows Forms)

Autres ressources

Gestion et déclenchement d'événements

Comment : compiler et exécuter un projet dans Visual Basic

L'environnement de développement intégré (IDE) de Visual Studio facilite la compilation d'un projet et l'exécution de l'application résultante.

Vous pouvez utiliser le débogueur intégré à l'IDE pour déboguer votre projet lors de son exécution. Pour plus d'informations sur le débogage, consultez Débogage de l'application Visual Basic et Modifier & Continuer (Visual Basic).

Remarque
Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour compiler et exécuter le projet actuel

- Appuyez sur F5 lorsque vous êtes dans l'environnement de développement intégré (IDE) de Visual Studio.

  L'IDE compile le projet et exécute l'application dans le débogueur de Visual Studio.

Pour compiler et exécuter le projet actuel à partir du menu

- Dans le menu **Déboguer** de l'IDE de Visual Studio, cliquez sur **Démarrer le débogage**.

  L'IDE compile le projet et exécute l'application dans le débogueur de Visual Studio.

Pour compiler et exécuter le projet actuel sans déboguer

- Appuyez sur CTRL+F5 dans l'IDE de Visual Studio.

  L'IDE compile le projet et exécute l'application.

Pour plus d'informations sur la compilation et l'exécution du code Visual Basic à l'aide du compilateur de ligne de commande, consultez Génération à partir de la ligne de commande (Visual Basic).

Voir aussi

Tâches

Comment : démarrer l'exécution

Concepts

Débogage de l'application Visual Basic

Autres ressources

Génération à partir de la ligne de commande (Visual Basic)

Modifier & Continuer (Visual Basic)

Sécurité et développement Visual Basic

Les développeurs Visual Basic doivent résoudre d'importants problèmes de sécurité lorsqu'ils commencent à utiliser le .NET Framework. Cette vue d'ensemble décrit les applications Windows et applications Web, l'implémentation, le débogage et les phases de déploiement du développement.

Vue d'ensemble

Visual Studio vous permet de contrôler la sécurité des applications en cours d'exécution. Même si vous avez le contrôle grâce au .NET Framework, vous devez réfléchir à votre programmation en tenant compte des problèmes de sécurité. Certains de ces problèmes doivent être résolus afin de pouvoir créer des applications conviviales et faciles à utiliser pour vos utilisateurs.

Il existe trois situations courantes où vous serez confronté à des problèmes de sécurité :

- Autorisations : l'utilisateur qui exécute votre application peut refuser des privilèges à celle-ci. Ce type de situation peut se produire si elle s'exécute à partir d'un emplacement qui, conformément aux spécifications de l'utilisateur, ne sera pas autorisé à accéder à certaines ressources système. Par exemple, l'utilisateur peut configurer le Common Language Runtime pour refuser des privilèges de fichier à toute application qui est stockée sur un lecteur réseau. Vous devez en tenir compte dans votre travail et écrire le code qui répond harmonieusement aux refus. Pour plus d'informations, consultez Stratégie de sécurité.
- Applications Web : vous devez empêcher les utilisateurs qui accèdent à vos applications Web depuis vos serveurs Web d'exécuter du code nuisible sur vos serveurs ou d'endommager les données qui s'y trouvent. Pour plus d'informations, consultez Méthodes de sécurité de base pour les applications Web (Visual Studio).
- Configuration de Visual Studio : selon la configuration de Visual Studio, votre serveur risque d'être plus ou moins exposé à des attaques par du code nuisible. Pour plus d'informations, consultez Autorisations de l'utilisateur et Visual Studio.

La sécurisation des ressources est un processus qui utilise plusieurs technologies et couvre l'ensemble du cycle de développement. En accordant un soin particulier à la conception, à l'implémentation, aux tests et au déploiement des applications, vous pouvez créer des applications très sécurisées. Les technologies de sécurité fournies par ASP.NET, le système d'exploitation et les navigateurs Web sont disponibles pour vous aider à sécuriser vos applications.

Pour plus d'informations, consultez Sécurité dans Visual Studio.

Voir aussi

Autres ressources

Indications de codage sécurisé
Sécurité des Windows Forms
Sécurité des sites Web ASP.NET (Visual Studio)
Sécurité dans les projets smart device
Exemples de sécurité Visual Basic

Concepts de base de la sécurité .Net Framework

La *sécurité d'accès du code* est la partie du .NET Framework qui contrôle l'accès aux ressources en contrôlant l'exécution du code. Cette fonctionnalité de sécurité est séparée de, et complète, la sécurité fournie par le système d'exploitation.

Fonctionnement

Lorsqu'un utilisateur lance votre application, une zone lui est assignée par le common language runtime .NET Framework. Les cinq zones sont les suivantes :

Zone	Description
Poste de travail	Le code de l'application réside sur l'ordinateur de l'utilisateur.
Intranet local	Le code de l'application s'exécute à partir d'un partage de fichiers sur l'intranet de l'utilisateur.
Internet	Le code de l'application s'exécute à partir d'Internet.
Sites de confiance	Ce sont des applications provenant de sites définis comme « De confiance » par Internet Explorer.
Sites non fiables	Ce sont des applications provenant de sites définis comme « Sensibles » par Internet Explorer.

L'assignation des trois premières zones, **Poste de travail**, **Intranet local** et **Internet**, dépend de l'endroit où se trouve le code. Vous pouvez substituer l'assignation en assignant des sites spécifiques aux groupes Sites de confiance ou Sites Non fiables dans Internet Explorer.

Chacune de ces zones a des autorisations d'accès spécifiques définies par un administrateur système. Le niveau de sécurité d'une zone peut être défini à confiance totale, confiance moyenne, confiance basse ou pas de confiance. Les niveaux de confiance définissent les ressources qui sont accessibles à l'application. La zone, avec d'autres preuves de sécurité, telle que l'éditeur, le nom fort, le site Web et l'URL du code, détermine les autorisations qui sont accordées au code au moment de l'exécution. (Pour plus d'informations sur les preuves de sécurité, consultez Preuve.) Vous n'avez aucun contrôle sur les paramètres de sécurité de l'ordinateur de l'utilisateur. Pourtant votre application doit fonctionner dans les paramètres qu'elle trouve lors de son exécution. Cela peut signifier que votre application se verra refuser l'accès à des ressources particulières. Par exemple, votre application peut avoir besoin d'écrire des données dans un fichier, mais le système de l'utilisateur refuse l'accès en écriture au moment de l'exécution en levant une exception.

Votre travail consiste à développer votre application de manière à gérer cette situation. Cela ne signifie pas nécessairement que vous ferez en sorte que votre application trouve une autre façon d'écrire les données. Cela signifie plutôt que votre application doit anticiper qu'elle ne pourra peut-être pas écrire les données puis répondre à une telle possibilité. Vous pouvez vouloir faire davantage appel à la gestion d'exceptions (Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic) ou à certains des objets dans l'espace de noms System.Security.Permissions pour rendre votre code plus fiable. Une brève description de ces méthodes est incluse dans une section ultérieure de cet article, « Développement pour les environnements de confiance partielle ».

Les niveaux de sécurité pour les zones sont définis à l'aide des outils d'administration installés en même temps que .NET Framework. Pour en savoir plus sur la définition des niveaux de sécurité pour les zones d'un ordinateur, consultez Outils d'administration.

Confiance totale

Les développeurs travaillent souvent dans un environnement de confiance totale. Ils gardent leur code source sur leurs disques durs, et ils testent leurs applications à partir de leurs ordinateurs de développement. Dans cet environnement de confiance totale, tout code que les développeurs compilent est autorisé à s'exécuter sur l'ordinateur local. Les exceptions de sécurité ne font pas surface, car l'ordinateur local est défini par défaut comme un environnement de confiance totale.

Confiance partielle

La confiance partielle décrit toute zone qui n'est pas une zone de confiance totale. Lorsqu'une application est déployée, elle peut se déplacer vers une nouvelle zone, qui ne lui accorde peut-être pas une confiance totale. Les deux scénarios les plus courants qui exécutent du code en confiance partielle sont les suivants :

- Exécution de code qui est téléchargé à partir d'Internet.

- Exécution de code qui réside sur un partage réseau (intranet).

Voici quelques exemples de ressources qui peuvent être refusées dans une zone de confiance partielle :

- Opérations d'E/S de fichier, notamment la lecture, l'écriture, la création, la suppression ou l'impression des fichiers.
- Composants système, tels que les valeurs du Registre et les variables d'environnement.
- Composants serveur, notamment les services d'annuaire, le Registre, les journaux d'événements, les compteurs de performance et les files d'attente de messages.

Qu'est-ce qui n'est pas autorisé en confiance partielle ? Ce n'est pas facile à déterminer. Chaque classe et chaque méthode de chaque classe du .NET Framework possède un attribut de sécurité qui définit le niveau de confiance nécessaire pour exécuter cette méthode. Cet attribut ne peut pas être accessible au moment de l'exécution à cause de ces fonctionnalités de sécurité. Le niveau de zone n'est pas un simple mappage du niveau de confiance aux attributs, mais une collection d'autorisations particulières données aux classes et méthodes particulières. Votre application ne pourra pas interroger simplement pour vérifier le niveau de confiance puis prédire quelles ressources ne sont pas disponibles. Vous pouvez déterminer si votre application s'exécute ou non en confiance totale. L'une des façons de procéder est décrite dans la section suivante, « Développement pour les environnements de confiance partielle ».

Développement pour les environnements de confiance partielle

Cette section présente brièvement la façon dont les problèmes de sécurité peuvent avoir une incidence sur le code que vous écrivez. Il n'y a aucune solution unique au développement pour les environnements de confiance partielle. Votre solution dépendra de l'application que vous écrivez. En outre, le niveau de confidentialité pouvant changer pendant l'exécution de votre application, vous ne pouvez pas tester simplement le niveau de confidentialité existant puis poursuivre.

Code d'exception de sécurité

La première étape du développement pour les zones de confiance partielle consiste à écrire du code qui reconnaît que les exceptions de sécurité feront surface. Prenons le code suivant :

```
Public Sub MakeABitmap()
    Dim b As Bitmap = New Bitmap(100, 100)
    ' Some code here to draw a nice picture in the bitmap
    b.Save("c:\PrettyPicture.bmp")
End Sub
```

Cette méthode s'exécute sans lever une exception si le projet et l'assembly de projet sont stockés sur le disque dur de votre ordinateur, et si vous êtes un membre du groupe Administrateurs sur votre ordinateur. Si vous déployez cette application sur votre intranet, une SecurityException peut être levée lorsque l'application tente d'enregistrer l'objet bitmap. Si vous n'avez pas de bloc Try...Catch...Finally, instruction (Visual Basic) autour de ce code, votre application se termine avec l'exception. Cela n'est pas probablement satisfaisant pour l'utilisateur. Si vous ajoutez le code de gestion d'exceptions, votre application peut :

- Prévenir l'utilisateur que l'application ne peut pas effectuer toutes les tâches qu'elle a besoin d'exécuter.
- Nettoyer tous les objets existants afin que le code qui s'exécute après le bloc catch n'échoue pas.

Vous pourriez modifier le code d'économie de bitmap comme indiqué ci-dessous. Le code ajouté prévient l'utilisateur que ce fichier n'a pas été enregistré à cause d'un refus de sécurité, et il sépare les défaillances de sécurité d'autres défaillances d'E/S de fichier, telles que les noms de fichiers inexacts. Cette méthode ne crée pas de failles de sécurité. L'utilisateur modifie la sécurité pour avoir confiance en votre application, sinon l'application ne s'exécute pas.

```
Public Sub MakeABitmap()
    Dim b As Bitmap = Nothing
    Try
        b = New Bitmap(100, 100)
        b.Save("c:\PrettyPicture.bmp")
    Catch ex As System.Security.SecurityException
        ' Let the user know the save won't work.
        MessageBox.Show("Permission to save the file was denied, " & _
           "and the bitmap was not saved.")
    Catch ex As System.Exception
        ' React to other exceptions here.
        MessageBox.Show("Unable to create and save the bitmap.")
    End Try
```

```
End Sub
```

Les classes, attributs et énumérations provenant de l'espace de noms **System.Security.Permissions** permettent une plus grande maîtrise des tâches de sécurité dans votre application. Si vous écrivez des bibliothèques qui peuvent être appelées à partir d'autres applications, votre bibliothèque doit vérifier les autorisations du code appelant. Par exemple, vous pourriez simplement ajouter l'attribut de niveau d'assembly suivant en haut de votre fichier de code, ou au fichier AssemblyInfo.vb ou AssemblyInfo.cs. Pour plus d'informations, consultez Définition des attributs d'assembly.

```
<Assembly: System.Security.Permissions.FileIOPermissionAttribute( _
    System.Security.Permissions.SecurityAction.RequestMinimum, _
    Write:="c:\PrettyPicture.bmp")>
```

L'exécution vérifie l'autorisation lors du chargement de l'assembly. Si l'exécution refuse l'autorisation demandée, le chargement de l'assembly échoue et une exception de sécurité est levée. Si vous ajoutez cet attribut à une application autonome, l'application peut ne pas s'exécuter. Si cet attribut apparaît dans une bibliothèque de classes, la bibliothèque peut ne pas se charger au moment de l'exécution. Vous devez alors ajouter un bloc try/catch au code qui appelle la bibliothèque de classes.

Vous pouvez également demander spécifiquement des autorisations de la part du runtime, indiquées ci-après à l'aide de la méthode `Demand`. Le runtime peut ensuite accorder ou refuser la demande. La demande est refusée par la levée d'une exception de sécurité. Vous pouvez réécrire le code comme suit pour demander explicitement l'autorisation d'écrire le fichier bitmap :

```
Public Sub MakeABitmap()
    Dim b As Bitmap = Nothing
    Dim filename As String = "c:\PrettyPicture.bmp"
    Dim permission As New System.Security.Permissions.FileIOPermission( _
        System.Security.Permissions.FileIOPermissionAccess.Write, _
        filename)

    Try
        permission.Demand()
    Catch ex As System.Security.SecurityException
        ' Let the user know the save won't work.
        MessageBox.Show("Permission to save the file was denied, " & _
           "and the bitmap was not saved.")
    Catch ex As System.Exception
        ' React to other exceptions here.
        MessageBox.Show("Other error.")
    End Try

    Try
        b = New Bitmap(100, 100)
        b.Save(filename)
    Catch ex As System.Exception
        MessageBox.Show("Unable to create and save the bitmap.")
    End Try
End Sub
```

Test

La deuxième étape du développement de zones de confiance partielle est le test dans plusieurs environnements, surtout provenant de votre intranet et d'Internet. Cela oblige la levée d'exceptions de sécurité. Un test important consiste à créer un compte d'utilisateur sur votre ordinateur local sans droits d'administrateur et à tenter d'exécuter votre application avec ce compte.

Voir aussi

Concepts

Introduction à la sécurité d'accès du code

Autres ressources

Sécurité d'accès du code

Espaces de noms de sécurité dans Visual Studio

Indications de codage sécurisé

Assistance au codage pour créer des applications sécurisées

En termes de sécurité, vous devez protéger vos ordinateurs de développement et vos serveurs d'attaques de code malveillant et de l'altération des données. Plusieurs mécanismes dans l'environnement de développement vous aident à sécuriser vos serveurs de développement :

Visual Studio Tools

Les groupes de comptes **Utilisateurs** sont ajoutés quand vous installez Visual Studio. Le groupe **Utilisateurs** détient les autorisations nécessaires liées aux fichiers, au partage et aux services IIS (Internet Information Services) pour créer et développer des applications Web sur un serveur. Il a également la capacité de déboguer des processus sur un ordinateur particulier, localement ou à distance. En tant que membre du groupe **Utilisateurs**, vous avez accès à la plupart des ressources. Pour plus d'informations, consultez Autorisations de l'utilisateur et Visual Studio.

Débogage

Nous vous conseillons de déboguer sur un ordinateur de test, plutôt que sur l'ordinateur de déploiement. Si vous devez déboguer sur un serveur de déploiement, installez uniquement le composant de débogage distant et désinstallez-le lorsque vous avez terminé. Mettez le serveur hors connexion lors du débogage. Pour plus d'informations, consultez Comment : installer le débogage distant. Pour des informations générales sur le débogage, consultez Sécurité du débogueur.

Déploiement

Pour la plupart des applications, il est suffisant que seul le .NET Framework soit installé sur le serveur. Si Visual Studio ou les composants serveur Visual Studio sont installés sur l'ordinateur de déploiement, le groupe **Utilisateurs** se trouve sur l'ordinateur de déploiement. De plus, vous pouvez également souhaiter désactiver la découverte dynamique.

Remarque

Nous vous conseillons fortement de ne pas installer Visual Studio 2005 sur votre serveur de déploiement. L'installation de Visual Studio ajoute des fichiers et des utilisateurs à votre système qui peuvent être exploités. Vous pouvez sécuriser un système sur lequel Visual Studio 2005 est installé, mais si vous n'avez pas besoin de Visual Studio 2005 sur le serveur de déploiement, nous vous conseillons de ne pas l'y installer.

La fonctionnalité Copier un projet de Visual Studio inclut l'option de déploiement d'une application avec un fichier de configuration (Web.config) qui est différente du fichier de configuration utilisé au cours du développement. Il est probable que, dans le fichier de développement, le débogage soit activé, lequel, s'il est déployé, permettrait aux utilisateurs d'examiner la pile des appels lorsqu'une exception est levée. Nous vous conseillons de déployer avec un fichier de configuration séparé qui n'autorise pas le débogage.

Pour plus d'informations, consultez Déploiement et sécurité ClickOnce et Comment : déboguer une application de confiance partielle.

Voir aussi

Référence

Les paramètres du proxy ne sont pas correctement configurés sur cet ordinateur pour la découverte Web.

Concepts

Sécurité et développement Visual Basic
Méthodes de sécurité de base pour les applications Web (Visual Studio)
Vue d'ensemble de la sécurité dans les Windows Forms

Programmation dans Visual Basic

Cette section décrit les tâches de programmation sur lesquelles vous voulez peut-être en savoir plus lorsque vous créez votre application Visual Basic.

Dans cette section

Accès à une application en cours d'exécution dans Visual Basic

Contient la documentation qui décrit comment contrôler le comportement de l'application au moment de l'exécution et obtenir des informations sur l'application à ce même moment.

Accès aux formulaires de l'application

Contient la documentation sur l'accès aux formulaires d'une application à l'aide des objets **My.Forms** et **My.Application**.

Accès aux ressources de l'application

Contient la documentation sur l'accès aux ressources d'une application à l'aide de l'objet **My.Resources**.

Accès aux paramètres d'application

Contient la documentation sur l'accès aux paramètres d'une application à l'aide de l'objet **My.Settings**.

Accès aux services Web d'une application

Contient la documentation qui décrit comment accéder aux services Web référencés par l'application à l'aide de l'objet **My.WebServices**.

Accès aux ressources de l'ordinateur

Contient la documentation qui décrit comment utiliser l'objet **My.Computer** pour accéder aux informations relatives à l'ordinateur sur lequel s'exécute une application et comment contrôler l'ordinateur.

Accès aux données de l'utilisateur

Contient la documentation sur les tâches que vous pouvez accomplir à l'aide de l'objet **My.User**.

Enregistrement d'informations provenant de l'application

Contient la documentation qui décrit comment enregistrer des informations à partir de votre application à l'aide des objets **My.Application.Log** et **My.Log** et comment étendre les fonctions de journalisation de l'application.

Traitement de lecteurs, de répertoires et de fichiers

Contient la documentation qui décrit comment accéder au système de fichiers à l'aide de l'objet **My.Computer.FileSystem**.

Mise à jour d'une application

Contient la documentation qui décrit comment mettre à jour des applications ClickOnce à l'aide de la propriété **My.Application.Deployment**

Voir aussi

Autres ressources

Développement d'applications avec Visual Basic

Accès aux ressources de l'ordinateur

L'objet **My.Computer** est l'un des trois objets centraux dans **My**, puisqu'il donne accès aux informations et aux fonctionnalités couramment utilisées. **My.Computer** fournit des méthodes, des propriétés et des événements pour accéder à l'ordinateur sur lequel l'application s'exécute. Il comprend les objets suivants :

- My.Computer.Audio, objet
- My.Computer.Clipboard, objet
- My.Computer.Clock, objet
- My.Computer.FileSystem, objet
- My.Computer.Info, objet
- My.Computer.Keyboard, objet
- My.Computer.Mouse, objet
- My.Computer.Network, objet
- My.Computer.Ports, objet
- My.Computer.Registry, objet

Dans cette section

Lecture de sons

Répertorie les tâches associées à **My.Computer.Audio**, par exemple lire un son en arrière-plan.

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

Répertorie les tâches associées à **My.Computer.Clipboard**, par exemple lire ou écrire des données dans le Presse-papiers.

Accès à l'horloge de l'ordinateur

Répertorie les tâches associées à **My.Computer.Clock**, par exemple déterminer l'heure locale du système.

Obtention d'informations sur l'ordinateur

Répertorie les tâches associées à **My.Computer.Info**, par exemple déterminer le nom complet d'un ordinateur ou obtenir des adresses IP.

Accès au clavier

Répertorie les tâches associées à **My.Computer.Keyboard**, par exemple déterminer si la touche VERR. MAJ est activée.

Accès à la souris

Répertorie les tâches associées à **My.Computer.Mouse**, par exemple déterminer si une souris est présente.

Exécution d'opérations de réseau

Répertorie les tâches associées à **My.Computer.Network**, par exemple transférer ou télécharger des fichiers.

Accès aux ports de l'ordinateur

Répertorie les tâches associées à **My.Computer.Ports**, par exemple afficher les ports série disponibles ou envoyer des chaînes aux ports série.

Lecture et écriture dans le Registre

Répertorie les tâches associées à **My.Computer.Registry**, par exemple lire ou écrire des données dans les clés de Registre.

Lecture de sons

L'objet **My.Computer.Audio** fournit des méthodes pour lire des sons.

Dans cette section

Comment : lire des sons et attendre qu'ils se terminent dans Visual Basic

Explique comment lire des sons à partir d'un fichier audio ou d'une ressource d'application et comment attendre la fin de leur exécution.

Comment : lire des sons dans Visual Basic

Explique comment lire un son en arrière-plan à partir d'un fichier audio ou d'une ressource d'application.

Comment : lire des sons en boucle dans Visual Basic

Explique comment lire un son en boucle continue à partir d'un fichier audio ou d'une ressource d'application.

Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic

Explique comment arrêter un son lu en arrière-plan.

Comment : lire des sons système dans Visual Basic

Explique comment lire un son système.

Référence

My.Computer.Audio, objet

Décrit l'objet **My.Computer.Audio**.

My.Computer.Audio.Play, méthode

Décrit la méthode **Play**.

My.Computer.Audio.PlaySystemSound, méthode

Décrit la méthode **PlaySystemSound**.

My.Computer.Audio.Stop, méthode

Décrit la méthode **Stop** qui est utilisée pour arrêter un son en cours de lecture.

Comment : lire des sons et attendre qu'ils se terminent dans Visual Basic

Cet exemple lit les sons d'un fichier son et d'une ressource d'application à l'aide de la méthode **My.Computer.Audio.Play** et attend qu'ils se terminent.

L'arrière-plan permet à l'application d'exécuter du code pendant la lecture des sons. La méthode **My.Computer.Audio.Play** permet à l'application de ne lire qu'un seul fond sonore à la fois. Lorsqu'elle lit un nouveau fond sonore, l'application arrête la lecture du fond sonore précédent.

La classe **My.Computer.Audio** fournit des méthodes et des propriétés qui peuvent être utilisées pour lire des fichiers audio.

Exemple

La méthode **My.Computer.Audio.Play** lit le son spécifié et attend qu'il se termine.

Vérifiez que le nom de fichier fait référence à un fichier son .wav présent sur votre système.

VB

```
Sub PlaySoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.WaitToComplete)
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Applications Windows Forms > Son**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

La méthode **My.Computer.Audio.Play** lit le son spécifié et attend qu'il se termine.

Vérifiez que les ressources de l'application incluent un fichier son .wav nommé `Waterfall`.

VB

```
Sub PlaySoundResource()
    My.Computer.Audio.Play(My.Resources.Waterfall, _
        AudioPlayMode.WaitToComplete)
End Sub
```

Compilation du code

Ces exemples de code ne peuvent s'exécuter que dans une application console ou Windows Forms. Pour plus d'informations, consultez My.Computer.Audio.Play, méthode.

Programmation fiable

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers son, pensez à stocker les fichiers comme ressources de l'application. Il est ensuite possible d'y accéder par l'intermédiaire de l'objet My.Resources.

Voir aussi

Tâches

Comment : lire des sons système dans Visual Basic
Comment : lire des sons dans Visual Basic

Référence

My.Computer.Audio.Play, méthode
AudioPlayMode, énumération
My.Computer.Audio, objet

Comment : lire des sons dans Visual Basic

Cet exemple lit un son en arrière-plan.

L'arrière-plan permet à l'application d'exécuter du code pendant la lecture des sons. La méthode **My.Computer.Audio.Play** permet à l'application de ne lire qu'un seul fond sonore à la fois. Lorsqu'elle lit un nouveau fond sonore, l'application arrête la lecture du fond sonore précédent. Si vous souhaitez lire un son et attendre qu'il se termine, consultez Comment : lire des sons et attendre qu'ils se terminent dans Visual Basic.

En général, lorsqu'une application lit un son en boucle, elle doit finir par l'arrêter. Pour plus d'informations, consultez Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic.

Exemple

La méthode **My.Computer.Audio.Play** lit un son en arrière-plan lorsque **PlayMode.Background** est spécifié.

Vérifiez que le nom de fichier fait référence à un fichier son .wav présent sur votre système.

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **PlayMode.Background** est spécifié.

Vérifiez que les ressources de l'application incluent un fichier son .wav nommé `Waterfall`.

VB

```
Sub PlayBackgroundSoundResource()
    My.Computer.Audio.Play(My.Resources.Waterfall, _
        AudioPlayMode.Background)
End Sub
```

Compilation du code

Ces exemples de code ne peuvent s'exécuter que dans une application console ou Windows Forms. Pour plus d'informations, consultez My.Computer.Audio.Play, méthode.

Programmation fiable

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers son, pensez à stocker les fichiers comme ressources de l'application. Il est ensuite possible d'y accéder avec l'objet My.Resources.

Voir aussi

Tâches

Comment : lire des sons système dans Visual Basic
Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic
Comment : lire des sons en boucle dans Visual Basic

Référence

My.Computer.Audio.Play, méthode
AudioPlayMode, énumération

Comment : lire des sons en boucle dans Visual Basic

Cet exemple lit un son en boucle en arrière-plan.

L'arrière-plan permet à l'application d'exécuter du code pendant la lecture des sons. Cela est particulièrement utile quand le son en cours de lecture ne doit pas bloquer l'exécution de l'application. La méthode **My.Computer.Audio.Play** permet à l'application de ne lire qu'un seul fond sonore à la fois. Lorsqu'elle lit un nouveau fond sonore, l'application arrête la lecture du fond sonore précédent.

En général, lorsqu'une application lit un son en boucle, elle doit finir par l'arrêter. Pour plus d'informations, consultez Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic.

Exemple

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **PlayMode.BackgroundLoop** est spécifié.

Vérifiez que le nom de fichier fait référence à un fichier son .wav présent sur votre système.

VB

```
Sub PlayLoopingBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.BackgroundLoop)
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Applications Windows Forms > Son**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **PlayMode.BackgroundLoop** est spécifié.

Vérifiez que les ressources de l'application incluent un fichier son .wav nommé `Waterfall`.

VB

```
Sub PlayLoopingBackgroundSoundResource()
    My.Computer.Audio.Play(My.Resources.Waterfall, _
          AudioPlayMode.BackgroundLoop)
End Sub
```

Compilation du code

Ces exemples de code ne peuvent s'exécuter que dans une application console ou Windows Forms. Pour plus d'informations, consultez My.Computer.Audio.Play, méthode.

Programmation fiable

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers son, pensez à stocker les fichiers comme ressources de l'application. Il est ensuite possible d'y accéder avec l'objet My.Resources.

Voir aussi

Tâches

Comment : lire des sons système dans Visual Basic
Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic
Comment : lire des sons dans Visual Basic

Référence

My.Computer.Audio.Play, méthode
AudioPlayMode, énumération

Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic

Cet exemple arrête un son en cours de lecture en arrière-plan.

L'arrière-plan permet à l'application d'exécuter du code pendant la lecture des sons. Pour plus d'informations, consultez Comment : lire des sons en boucle dans Visual Basic et Comment : lire des sons dans Visual Basic.

En général, lorsqu'une application lit un son en boucle, elle doit l'arrêter à un certain point.

Exemple

Utilisez la méthode **My.Computer.Audio.Stop** pour arrêter la lecture en cours du fonds sonore ou du son en boucle de l'application.

VB

```
Sub StopBackgroundSound()
    My.Computer.Audio.Stop()
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Applications Windows Forms > Son**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple de code ne peut s'exécuter que dans une application console ou Windows Forms. Pour plus d'informations, consultez My.Computer.Audio.Stop, méthode.

Voir aussi

Tâches

Comment : lire des sons dans Visual Basic

Comment : déterminer si un fichier existe dans Visual Basic

Référence

My.Computer.Audio.Stop, méthode

Autres ressources

Lecture de sons

Comment : lire des sons système dans Visual Basic

Cet exemple utilise la méthode **My.Computer.Audio.PlaySystemSound** pour lire un son système.

La méthode **My.Computer.Audio.PlaySystemSound** prend comme paramètre l'un des membres partagés de la classe SystemSound.

Exemple

Utilisez la méthode **My.Computer.Audio.PlaySystemSound** pour lire le son système spécifié.

Le son système Asterisk indique généralement des erreurs. Pour plus d'informations, consultez **SystemSound**.

VB

```
Sub PlaySystemSound()
    My.Computer.Audio.PlaySystemSound( _
        System.Media.SystemSounds.Asterisk)
End Sub
```

Compilation du code

Cet exemple de code ne peut s'exécuter que dans une application console ou Windows Forms. Pour plus d'informations, consultez My.Computer.Audio.PlaySystemSound, méthode.

Voir aussi

Tâches

Comment : lire des sons dans Visual Basic
Comment : lire des sons en boucle dans Visual Basic
Comment : déterminer si un fichier existe dans Visual Basic

Référence

My.Computer.Audio.Stop, méthode
System.Media.SystemSound

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

L'objet **My.Computer.Clipboard** fournit des méthodes et des propriétés qui peuvent être utilisées pour interagir avec le Presse-papiers. Elles permettent à l'utilisateur d'écrire des données dans le Presse-papiers ou d'obtenir des données à partir du Presse-papiers, et de vérifier si les données d'un format spécifique sont présentes dans le Presse-papiers. Le Presse-papiers est un ensemble de fonctions et de messages qui permettent aux applications de transférer des données. Comme toutes les applications ont accès au Presse-papiers, les données peuvent être transférées facilement entre elles.

Dans cette section

Comment : effacer le Presse-papiers en Visual Basic

Explique comment effacer le Presse-papiers.

Comment : lire le Presse-papiers en Visual Basic

Explique comment lire des données contenues dans le Presse-papiers.

Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Explique comment déterminer quel type de données est enregistré dans le Presse-papiers.

Comment : écrire dans le Presse-papiers en Visual Basic

Explique comment écrire des données dans le Presse-papiers.

Comment : récupérer une image du Presse-papiers en Visual Basic

Explique comment récupérer une image du Presse-papiers.

Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic

Explique comment enregistrer des données audio dans le Presse-papiers.

Référence

My.Computer.Clipboard, objet

Décrit l'objet **My.Computer.Clipboard**, ses méthodes et ses propriétés.

Rubriques connexes

Sérialisation

Décrit la sérialisation. Une classe doit être sérialisable pour pouvoir être placée dans le Presse-papiers.

Comment : effacer le Presse-papiers en Visual Basic

La méthode My.Computer.Clipboard.Clear efface le Presse-papiers. Comme le Presse-papiers est partagé par d'autres processus, si vous l'effacez, cela peut avoir un impact sur ces processus.

Pour effacer le Presse-papiers

- Utilisez la méthode **Clear**, comme suit.

  VB

```
My.Computer.Clipboard.Clear()
```

Voir aussi

Tâches

Comment : lire le Presse-papiers en Visual Basic
Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic
Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic
Comment : écrire dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard.Clear, méthode

Comment : lire le Presse-papiers en Visual Basic

Le Presse-papiers peut être utilisé pour stocker des données, telles que du texte et des images. Comme il est partagé par tous les processus actifs, il peut servir à transférer des données entre ces processus. L'objet **My.Computer.Clipboard** vous permet d'accéder facilement au Presse-papiers et de lire et écrire dedans. Les méthodes **GetText**, **GetImage**, **GetData**, **GetAudioStream** et **GetFileDropDownList** vous permettent de spécifier le type de données que vous voulez lire dans le Presse-papiers.

Si les données ne peuvent pas être récupérées du Presse-papiers, une exception ExternalException est levée.

Pour lire du texte contenu dans le Presse-papiers et l'afficher

- Utilisez la méthode **My.Computer.Clipboard.GetText** pour lire le texte. Le code suivant permet de lire le texte et de l'afficher dans une boîte de message. Pour que l'exemple s'exécute correctement, du texte doit être stocké dans le Presse-papiers.

  VB

  ```
  MsgBox(My.Computer.Clipboard.GetText())
  ```

  Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Applications Windows Forms > Presse-papiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Pour lire une image contenue dans le Presse-papiers

- Utilisez la méthode **My.Computer.Clipboard.GetImage** pour lire une image dans le Presse-papiers. Le code suivant lit une image contenue dans le Presse-papiers et l'assigne à la propriété `Image` de `Button1`. Il doit exister dans le Presse-papiers une image stockée et un Bouton nommé `Button1` pour que l'exemple s'exécute correctement.

  VB

  ```
  Button1.Image = My.Computer.Clipboard.GetImage()
  ```

  Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Applications Windows Forms > Presse-papiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Voir aussi

Tâches

Comment : écrire dans le Presse-papiers en Visual Basic
Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, membres de l'objet
My.Computer.Clipboard, objet

Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Le Presse-papiers peut être utilisé pour stocker des données, telles que du texte et des images. Comme il est partagé par tous les processus, il peut servir à transférer des données entre ces processus. L'objet **My.Computer.Clipboard** vous permet d'accéder facilement au Presse-papiers et de lire et écrire dedans.

Les données dans le Presse-papiers peuvent avoir plusieurs formats différents : texte, fichier audio ou image. Pour déterminer quel type de fichier est dans le Presse-papiers, vous pouvez utiliser des méthodes telles que **ContainsAudio**, **ContainsFileDropList**, **ContainsImage** et **ContainsText**. La méthode **ContainsData** peut être utilisée pour vérifier un format personnalisé.

Pour déterminer si le Presse-papiers contient une image

- Utilisez la fonction **ContainsImage** pour déterminer si les données dans le Presse-papiers correspondent à une image. Le code suivant vérifie si les données correspondent à une image et indique le résultat.

  VB

```
If My.Computer.Clipboard.ContainsImage() = True Then
    MsgBox("Clipboard contains an image.")
Else
    MsgBox("Clipboard does not contain an image.")
End If
```

Voir aussi

Tâches

Comment : lire le Presse-papiers en Visual Basic
Comment : écrire dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.ContainsAudio, méthode
My.Computer.Clipboard.ContainsFileDropList, méthode
My.Computer.Clipboard.ContainsImage, méthode
My.Computer.Clipboard.ContainsText, méthode
My.Computer.Clipboard.ContainsData, méthode

Comment : écrire dans le Presse-papiers en Visual Basic

Le Presse-papiers peut être utilisé pour stocker des données, telles que du texte et des images. Comme il est partagé par tous les processus, il peut servir à transférer des données entre ces processus. L'objet **My.Computer.Clipboard** vous permet d'accéder facilement au Presse-papiers et de lire et écrire dedans. Les méthodes **SetAudio**, **SetData**, **SetFileDropDownList**, **SetImage** et **SetText** vous permettent de placer des données dans le Presse-papiers.

Remarque de sécurité
Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Pour écrire du texte dans le Presse-papiers

- Utilisez la méthode **My.Computer.Clipboard.SetText** pour écrire du texte dans le Presse-papiers. Le code suivant écrit la chaîne "This is a test string" dans le Presse-papiers.

  VB

  ```
  My.Computer.Clipboard.SetText("This is a test string.")
  ```

Pour écrire du texte dans le Presse-papiers dans un format spécifique

- Utilisez la méthode **My.Computer.Clipboard.SetText** pour écrire du texte dans le Presse-papiers, notamment le type de TextDataFormat. Le code suivant écrit la chaîne "This is a test string" au format RTF dans le Presse-papiers.

  VB

  ```
  My.Computer.Clipboard.SetText("This is a test string.", _
  System.Windows.Forms.TextDataFormat.Rtf)
  ```

Pour écrire des données dans le Presse-papiers

- Utilisez la méthode **My.Computer.Clipboard.SetData** pour écrire des données dans le Presse-papiers. Cet exemple écrit le DataObject `dataChunk` dans le Presse-papiers au format personnalisé `specialFormat`.

  VB

  ```
  My.Computer.Clipboard.SetData("specialFormat", dataChunk)
  ```

Voir aussi

Tâches

Comment : lire le Presse-papiers en Visual Basic
Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.SetText, méthode
My.Computer.Clipboard.SetData, méthode
My.Computer.Clipboard.SetDataObject, méthode
TextDataFormat

Comment : récupérer une image du Presse-papiers en Visual Basic

La méthode **My.Computer.Clipboard.GetImage** vous permet de récupérer une image du Presse-papiers.

Les éléments placés dans le Presse-papiers seront persistants même après la fermeture de l'application.

Pour récupérer une image du Presse-papiers

- Utilisez la méthode **My.Computer.Clipboard.GetImage** pour récupérer une image du Presse-papiers. Cet exemple vérifie si une image est bien présente dans le Presse-papiers avant de la récupérer et de l'assigner à `PictureBox1`.

  VB

```
If My.Computer.Clipboard.ContainsImage() Then
  Dim grabpicture As System.Drawing.Image
  grabpicture = My.Computer.Clipboard.GetImage()
  picturebox1.Image = grabpicture
End If
```

Voir aussi

Tâches

Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard, membres de l'objet
My.Computer.Clipboard.ContainsImage, méthode
My.Computer.Clipboard.GetImage, méthode

Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic

La méthode **My.Computer.Clipboard.SetAudio** peut être utilisée pour enregistrer des données audio dans le Presse-papiers.

Pour enregistrer des données audio dans le Presse-papiers

- Utilisez la méthode **My.Computer.Clipboard.SetAudio** pour écrire les données audio dans le Presse-papiers. Dans cet exemple, le tableau d'octets `musicReader` est créé, le fichier `cool.wav` de ce tableau est lu, puis enregistré dans le Presse-papiers.

  VB

```
Dim musicReader As Byte()
musicReader = My.Computer.FileSystem.ReadAllBytes("cool.wav")
My.Computer.Clipboard.SetAudio(musicReader)
```

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.SetAudio, méthode
My.Computer.Clipboard.GetAudioStream, méthode
My.Computer.Clipboard.ContainsAudio, méthode

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

Accès à l'horloge de l'ordinateur

L'objet **My.Computer.Clock** fournit des propriétés pour accéder à l'heure locale actuelle et au temps UTC (Universal Coordinated Time), équivalent à l'heure GMT (Greenwich Mean Time), à partir de l'horloge système.

Notes

Ce tableau répertorie les tâches courantes qui impliquent l'objet **My.Computer.Clock** et indique les rubriques qui expliquent chacune de ces tâches.

Pour	Reportez-vous à
Déterminer l'heure GMT	My.Computer.Clock.GmtTime, propriété
Déterminer l'heure locale du système	My.Computer.Clock.LocalTime, propriété
Déterminer le nombre de graduations qui se sont écoulées depuis le démarrage de l'ordinateur	My.Computer.Clock.TickCount, propriété

Voir aussi

Référence

My.Computer.Clock, objet
Membres de l'objet My.Computer.Clock
System.DateTime

Concepts

Mise en forme de la date et de l'heure pour une culture spécifique

Obtention d'informations sur l'ordinateur

L'objet **My.Computer.Info** fournit des propriétés pour obtenir des informations concernant la mémoire, les assemblys chargés, le nom et le système d'exploitation de l'ordinateur.

Notes

Ce tableau répertorie les tâches couramment exécutées à l'aide de l'objet **My.Computer.Info** et indique les rubriques qui expliquent comment les exécuter.

Pour	Reportez-vous à
Déterminer combien d'espace d'adressage virtuel est disponible sur l'ordinateur sur lequel l'application est installée	My.Computer.Info.TotalVirtualMemory, propriété
Déterminer le type de plate-forme de l'ordinateur sur lequel l'application s'exécute	My.Computer.Info.OSPlatform, propriété
Déterminer le système d'exploitation de l'ordinateur sur lequel l'application s'exécute	My.Computer.Info.OSFullName, propriété
Déterminer les Service Packs installés sur l'ordinateur sur lequel l'application s'exécute	My.Computer.Info.OSVersion, propriété
Déterminer la propriété **UICulture** installée sur l'ordinateur sur lequel l'application s'exécute	My.Computer.Info.InstalledUICulture, propriété

Voir aussi

Référence

My.Computer.Info, objet
Membres de l'objet My.Computer.Info

Accès au clavier

L'objet **My.Computer.Keyboard** fournit des propriétés pour accéder à l'état actuel du clavier, par exemple pour savoir quelles touches sont actuellement utilisées, et fournit une méthode pour envoyer des séquences de touches à la fenêtre active.

Tâches

Ce tableau répertorie les tâches associées à l'objet **My.Computer.Keyboard** et indique les rubriques qui expliquent comment accomplir chaque tâche.

Pour	Reportez-vous à
Déterminer si la touche VERR. MAJ est activée	Comment : déterminer si la touche Verr. Maj. est activée dans Visual Basic
Déterminer si la touche MAJ est enfoncée	My.Computer.Keyboard.ShiftKeyDown, propriété
Déterminer si la touche ALT est enfoncée	My.Computer.Keyboard.AltKeyDown, propriété
Déterminer si la touche CTRL est enfoncée	My.Computer.Keyboard.CtrlKeyDown, propriété
Déterminer si la touche VERR. NUM est activée	My.Computer.Keyboard.NumLock, propriété
Déterminer si la touche ARRÊT DÉFIL est activée	My.Computer.Keyboard.ScrollLock, propriété
Démarrer une application et lui envoyer des séquences de touches	Comment : démarrer une application et envoyer des séquences de touches (Visual Basic)

Voir aussi

Référence

My.Computer.Keyboard, objet
System.Windows.Forms.Keys

Comment : déterminer si la touche Verr. Maj. est activée dans Visual Basic

L'objet **My.Computer.Keyboard** peut être utilisé pour déterminer l'état actuel du clavier, par exemple pour savoir si la touche VERR. NUM ou VERR. MAJ est activée.

Pour déterminer si la touche VERR. MAJ est activée

- Pour déterminer si la touche VERR. MAJ est activée ou non, utilisez la propriété **My.Computer.Keyboard.CapsLock**. Le code suivant affiche un message sur l'état de la touche VERR. MAJ.

  VB

```
If My.Computer.Keyboard.CapsLock Then
    MsgBox("CAPS LOCK is on")
Else
    MsgBox("CAPS LOCK is off")
End If
```

Voir aussi

Référence

My.Computer.Keyboard, objet

Concepts

Accès au clavier

Comment : démarrer une application et envoyer des séquences de touches (Visual Basic)

Cet exemple utilise la fonction **Shell** pour démarrer l'application de calculatrice, puis multiplie deux nombres en envoyant des séquences de touches à l'aide de la méthode **My.Computer.Keyboard.SendKeys**.

Exemple

VB

```
Dim ProcID As Integer
' Start the Calculator application, and store the process id.
ProcID = Shell("CALC.EXE", AppWinStyle.NormalFocus)
' Activate the Calculator application.
AppActivate(ProcID)
' Send the keystrokes to the Calculator application.
My.Computer.Keyboard.SendKeys("22", True)
My.Computer.Keyboard.SendKeys("*", True)
My.Computer.Keyboard.SendKeys("44", True)
My.Computer.Keyboard.SendKeys("=", True)
' The result is 22 * 44 = 968.
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Applications Windows Forms > Formulaires**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Une exception ArgumentException est levée si une application avec l'identificateur de processus demandé ne peut pas être trouvée.

Sécurité

L'appel à la fonction **Shell** requiert une confiance totale (classe SecurityException).

Voir aussi

Référence

My.Computer.Keyboard.SendKeys, méthode
Shell, fonction
AppActivate, fonction

Accès à la souris

L'objet **My.Computer.Mouse** permet de rechercher des informations à propos de la souris d'un ordinateur, par exemple savoir si une souris existe, connaître le nombre de boutons qu'elle possède ou les détails concernant la roulette de la souris.

Notes

Ce tableau répertorie les tâches associées à l'objet **My.Computer.Mouse** et indique les rubriques qui expliquent comment les accomplir.

Pour	Reportez-vous à
Déterminer si la souris a une roulette de défilement.	My.Computer.Mouse.WheelExists, propriété
Déterminer si les boutons gauche et droit de la souris ont été inversés.	My.Computer.Mouse.ButtonsSwapped, propriété
Définir la longueur du défilement à l'écran lorsque la roulette de la souris est tournée d'un cran.	My.Computer.Mouse.WheelScrollLines, propriété

Voir aussi

Référence

My.Computer.Mouse, objet

Membres de l'objet My.Computer.Mouse

Exécution d'opérations de réseau

Les tableaux suivants répertorient les tâches associées à l'objet **My.Computer.Network**.

Dans cette section

Comment : transférer un fichier dans Visual Basic

Montre comment télécharger un fichier et le stocker dans un emplacement distant à l'aide de **My.Computer.Network**.

Comment : télécharger un fichier dans Visual Basic

Montre comment télécharger un fichier à partir d'un emplacement distant à l'aide de **My.Computer.Network**.

Comment : vérifier l'état de la connexion dans Visual Basic

Indique comment déterminer si la connexion réseau de l'ordinateur est active.

Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Indique comment utiliser la méthode **Ping** pour déterminer si un ordinateur distant ou hôte est disponible.

Référence

My.Computer.Network, objet

Répertorie les méthodes, les propriétés et les événements permettant d'utiliser le réseau.

My.Computer.Network.DownloadFile, méthode

Décrit la méthode **DownloadFile**.

My.Computer.Network.Ping, méthode

Décrit la méthode **Ping**.

My.Computer.Network.UploadFile, méthode

Décrit la méthode **UploadFile**.

My.Computer.Network.IsAvailable, propriété

Décrit la propriété **IsAvailable**.

Custom

Décrit l'événement **NetworkAvailable**.

Comment : transférer un fichier dans Visual Basic

La méthode My.Computer.Network.UploadFile peut être utilisée pour transférer un fichier et le stocker dans un emplacement distant. Si le paramètre *ShowUI* a la valeur **True**, une boîte de dialogue s'affiche pour indiquer la progression du téléchargement et permet aux utilisateurs d'annuler l'opération.

Pour transférer un fichier

- Utilisez la méthode **UploadFile** pour transférer un fichier, en spécifiant l'emplacement du fichier source et l'emplacement de répertoire cible sous forme de chaîne ou d'URI. Cet exemple transfère le fichier `Order.txt` vers `http://www.cohowinery.com/uploads.aspx`

  VB

```
My.Computer.Network.UploadFile( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx")
```

Pour transférer un fichier et afficher la progression de l'opération

- Utilisez la méthode **UploadFile** pour transférer un fichier en spécifiant l'emplacement du fichier source et l'emplacement de répertoire cible sous forme de chaîne ou d'URI. Cet exemple transfère le fichier `Order.txt` vers `http://www.cohowinery.com/uploads.aspx` sans fournir un nom d'utilisateur ou un mot de passe, affiche la progression du transfert et a un intervalle de délai d'attente de 500 millisecondes.

  VB

```
My.Computer.Network.UploadFile( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx", "", "", True, 500)
```

Pour transférer un fichier en fournissant un nom d'utilisateur et un mot de passe

- Utilisez la méthode **UploadFile** pour transférer un fichier en spécifiant l'emplacement du fichier source et l'emplacement du répertoire cible sous forme de chaîne ou d'URI, puis en spécifiant le nom d'utilisateur et le mot de passe. Cet exemple transfère le fichier `Order.txt` vers `http://www.cohowinery.com/uploads.aspx`en fournissant le nom d'utilisateur `anonymous` et un mot de passe vide.

  VB

```
My.Computer.Network.UploadFile( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx", "anonymous", "")
```

Programmation fiable

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès du fichier local n'est pas valide (ArgumentException).
- L'authentification a échoué (SecurityException).
- La connexion a expiré (TimeoutException).

Voir aussi

Tâches

Comment : télécharger un fichier dans Visual Basic

Comment : déterminer si un ordinateur distant est disponible dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.Network, objet
Membres de l'objet My.Computer.Network
My.Computer.Network.UploadFile, méthode

Comment : télécharger un fichier dans Visual Basic

La My.Computer.Network.DownloadFile, méthode peut être utilisée pour télécharger un fichier distant et le stocker dans un emplacement spécifique. Si le paramètre *ShowUI* a la valeur **True**, une boîte de dialogue s'affiche pour indiquer la progression du téléchargement et permettre aux utilisateurs d'annuler l'opération. Par défaut, les fichiers existants portant le même nom ne sont pas remplacés ; si vous souhaitez le faire, affectez le paramètre *overwrite* à **True**.

Les conditions ci-dessous peuvent générer une exception :

- Le nom du lecteur est incorrect (ArgumentException).
- L'authentification nécessaire n'a pas été fournie (UnauthorizedAccessException ou SecurityException).
- Le serveur ne répond pas dans le *connectionTimeout* spécifié (TimeoutException).
- La demande est refusée par le site Web (WebException).

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concern e les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation d e paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Remarque de sécurité

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut qu'un fichier nommé Form 1.vb ne soit pas un fichier source Visual Basic. Vérifiez toutes les entrées avant d'utiliser les données dans votre application. L e fichier n'a peut-être pas le contenu attendu, et les méthodes utilisées pour lire le fichier peuvent échouer.

Pour télécharger un fichier

- Utilisez la méthode **DownloadFile** pour télécharger le fichier, en spécifiant l'emplacement du fichier cible sous forme de chaîne ou d'URI et en spécifiant l'emplacement dans lequel stocker le fichier. Cet exemple télécharge le fichier `WineList.txt` à partir de `http://www.cohowinery.com/downloads` et l'enregistre dans `C:\Documents and Settings\All Users\Documents`.

  VB

```
My.Computer.Network.DownloadFile _
    ("http://www.cohowinery.com/downloads/WineList.txt", _
    "C:\Documents and Settings\All Users\Documents\WineList.txt")
```

Pour télécharger un fichier en spécifiant un délai d'attente

- Utilisez la méthode **DownloadFile** pour télécharger le fichier en spécifiant l'emplacement du fichier cible sous forme de chaîne ou d'URI, en spécifiant l'emplacement dans lequel stocker le fichier et en spécifiant le délai d'attente en millisecondes (la valeur par défaut étant de 1000). Cet exemple télécharge le fichier `WineList.txt` à partir du site `http://www.cohowinery.com/downloads` et le sauvegarde dans `C:\Documents and Settings\All Users\Documents` en spécifiant un délai d'attente de 500 millisecondes :

  VB

```
My.Computer.Network.DownloadFile _
    ("http://www.cohowinery.com/downloads/WineList.txt", _
    "C:\Documents and Settings\All Users\Documents\WineList.txt", False, 500)
```

Pour télécharger un fichier en fournissant un nom d'utilisateur et un mot de passe

- Utilisez la méthode **DownLoadFile** pour télécharger le fichier, en spécifiant l'emplacement du fichier cible sous forme de chaîne ou d'URI et en spécifiant l'emplacement dans lequel stocker le fichier, le nom d'utilisateur et le mot de passe. Cet exemple télécharge le fichier `WineList.txt` à partir du site `http://www.cohowinery.com/downloads` et le sauvegarde dans `C:\Documents and Settings\All Users\Documents`, avec le nom d'utilisateur `anonymous` et un mot de passe vide.

  VB

```
My.Computer.Network.DownloadFile _
    ("http://www.cohowinery.com/downloads/WineList.txt", _
    "C:\Documents and Settings\All Users\Documents\WineList.txt", "anonymous", "")
```

Remarque de sécurité
Le protocole FTP utilisé par la méthode **DownLoadFile** envoie les informations, y compris les mots de passe, en texte brut, et ne doit pas être utilisé pour transmettre des informations sensibles.

Voir aussi

Tâches

Comment : transférer un fichier dans Visual Basic

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.Network, objet

Membres de l'objet My.Computer.Network

My.Computer.Network.DownloadFile, méthode

Comment : vérifier l'état de la connexion dans Visual Basic

La propriété My.Computer.Network.IsAvailable peut être utilisée pour déterminer si l'ordinateur a un réseau actif ou une connexion Internet.

> **Remarque**
>
> Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour vérifier si un ordinateur a une connexion active

- Déterminez si la propriété **IsAvailable** a la valeur **True** ou **False**. Le code suivant vérifie l'état de la propriété et le signale :

  VB

  ```
  If My.Computer.Network.IsAvailable = True Then
      MsgBox("Computer is connected.")
  Else
      MsgBox("Computer is not connected.")
  End If
  ```

  Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Connectivité et réseau**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Voir aussi

Tâches

Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Référence

My.Computer.Network, objet

My.Computer.Network.IsAvailable, propriété

Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Vous pouvez utiliser la méthode My.Computer.Network.Ping pour déterminer si un ordinateur distant ou un hôte est disponible. Le serveur peut être spécifié par URL, nom d'ordinateur ou adresse IP. N'incluez pas `http://` lors de la spécification d'une URL.

La méthode **Ping** n'est pas une méthode infaillible pour déterminer la disponibilité d'un ordinateur distant : le port PING de l'ordinateur cible peut être désactivé, ou la demande PING peut être bloquée par un pare-feu ou un routeur.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour exécuter la commande PING sur un serveur

- Déterminez si la méthode **Ping** retourne **True**. Cet exemple indique si la commande PING peut être exécutée sur le serveur ou non, en déterminant si la méthode **Ping** a retourné **True**. Remplacez `198.01.01.01` par l'adresse IP, l'URL ou le nom d'ordinateur du serveur sur lequel exécuter la commande PING

  VB

```
If My.Computer.Network.Ping("198.01.01.01") Then
  MsgBox("Server pinged successfully.")
Else
  MsgBox("Ping request timed out.")
End If
```

Pour exécuter la commande PING sur un serveur et spécifier un délai

- Déterminez si la méthode **Ping** retourne **True** en spécifiant l'intervalle de délai en millisecondes. Si aucun délai n'est spécifié, `500` est utilisé comme valeur par défaut. Cet exemple indique si la commande PING peut être exécutée sur le serveur ou non en déterminant si la méthode **Ping** a retourné **True**, et spécifie un intervalle de délai de 1000 millisecondes. Remplacez `www.cohowinery.com` par l'adresse IP, l'URL ou le nom d'ordinateur du serveur sur lequel exécuter la commande PING.

  VB

```
If My.Computer.Network.Ping("www.cohowinery.com", 1000) Then
  MsgBox("Server pinged successfully.")
Else
  MsgBox("Ping request timed out.")
End If
```

Voir aussi

Tâches

Comment : vérifier l'état de la connexion dans Visual Basic

Référence

My.Computer.Network, objet

Membres de l'objet My.Computer.Network

My.Computer.Network.Ping, méthode

Accès aux ports de l'ordinateur

L'objet **My.Computer.Ports** fournit une propriété et une méthode pour accéder aux ports série de l'ordinateur.

Dans cette section

Comment : afficher les ports série disponibles dans Visual Basic

Explique comment afficher les ports série disponibles.

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic

Explique comment utiliser un modem connecté au port série d'un ordinateur.

Comment : envoyer des chaînes aux ports série dans Visual Basic

Explique comment envoyer une chaîne au port série d'un ordinateur.

Comment : recevoir des chaînes des ports série dans Visual Basic

Explique comment recevoir une chaîne du port série d'un ordinateur.

Référence

My.Computer.Ports, objet

Décrit l'objet **My.Computer.Ports** et ses membres.

My.Computer.Ports.SerialPortNames, propriété

Décrit la propriété **SerialPortNames** qui obtient une collection des noms des ports série disponibles sur l'ordinateur.

My.Computer.Ports.OpenSerialPort, méthode

Décrit la méthode **OpenSerialPort** qui crée et ouvre un objet System.IO.Ports.SerialPort.

Rubriques connexes

SerialPort

Décrit la classe **SerialPort** du .NET Framework.

Comment : afficher les ports série disponibles dans Visual Basic

Cette rubrique décrit comment utiliser **My.Computer.Ports** pour afficher les ports série de l'ordinateur disponibles en Visual Basic.

Pour permettre à l'utilisateur de sélectionner le port à utiliser, les noms des ports série sont placés dans un contrôle ListBox.

Exemple

Cet exemple fait une boucle sur toutes les chaînes que la propriété **My.Computer.Ports.SerialPortNames** retourne. Ces chaînes constituent les noms des ports série disponibles sur l'ordinateur.

En général, l'utilisateur sélectionne le port série que l'application doit utiliser dans la liste de ports disponibles. Dans cet exemple, les noms de ports série sont stockés dans un contrôle **ListBox**. Pour plus d'informations, consultez ListBox, contrôle (Windows Forms).

VB

```
Sub GetSerialPortNames()
    ' Show all available COM ports.
    For Each sp As String In My.Computer.Ports.SerialPortNames
        ListBox1.Items.Add(sp)
    Next
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Connectivité et réseau**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite les éléments suivants :

- Une référence de projet à System.Windows.Forms.dll.
- L'accès aux membres de l'espace de noms System.Windows.Forms. L'ajout d'une instruction **Imports** si vos noms de membres ne sont pas qualifiés complets dans votre code. Pour plus d'informations, consultez Imports, instruction.
- Que votre formulaire ait un contrôle **ListBox** nommé `ListBox1`.

Programmation fiable

Vous ne devez pas utiliser le contrôle **ListBox** pour afficher les noms de ports série disponibles. À la place, vous pouvez utiliser un ComboBox ou un autre contrôle. Si l'application n'a pas besoin d'une réponse de l'utilisateur, vous pouvez utiliser un contrôle TextBox pour afficher les informations.

Remarque

Les noms de ports retournés par **My.Computer.Ports.SerialPortNames** peuvent être inexacts lorsqu'ils sont exécutés sous Windows 98. Pour empêcher des erreurs d'application, utilisez la gestion des exceptions, telle que l'instruction **Try...Catch...Finally** ou l'instruction **Using**, lors de l'utilisation des noms de ports pour ouvrir des ports.

Voir aussi

Tâches

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Comment : envoyer des chaînes aux ports série dans Visual Basic
Comment : recevoir des chaînes des ports série dans Visual Basic

Référence

My.Computer.Ports, objet

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic

Cette rubrique décrit comment utiliser **My.Computer.Ports** pour utiliser un modem dans Visual Basic.

En général, le modem est connecté à l'un des ports série de l'ordinateur. Pour que votre application communique avec le modem, elle doit envoyer des commandes au port série approprié.

Pour utiliser un modem

1. Déterminez le port série auquel le modem est connecté. Cet exemple suppose que le modem est connecté à COM1.
2. Utilisez la méthode **My.Computer.Ports.OpenSerialPort** pour obtenir une référence au port. Pour plus d'informations, consultez My.Computer.Ports.OpenSerialPort, méthode.

   Le bloc **Using** permet à l'application de fermer le port série même s'il génère une exception. Tout le code qui manipule le port série doit apparaître dans ce bloc ou dans un bloc **Try...Catch...Finally**.

   VB
   ```
   Using com1 As IO.Ports.SerialPort = _
           My.Computer.Ports.OpenSerialPort("COM1", 9600)
   End Using
   ```
3. Définissez la propriété **DtrEnable** pour indiquer que l'ordinateur est prêt à accepter une transmission entrante du modem.

   VB
   ```
   com1.DtrEnable = True
   ```
4. Envoyez la commande de numérotation et le numéro de téléphone au modem via le port série à l'aide de la méthode Write.

   VB
   ```
   com1.Write("ATDT 555-0100" & vbCrLf)
   ```

Exemple

VB
```
Sub DialModem()
    ' Dial a number via an attached modem on COM1.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1", 9600)
        com1.DtrEnable = True
        com1.Write("ATDT 555-0100" & vbCrLf)
        ' Insert code to transfer data to and from the modem.
    End Using
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Connectivité et réseau**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple requiert une référence à l'espace de noms System.

Programmation fiable

Cet exemple suppose que le modem est connecté à COM1. Nous recommandons que votre code permette à l'utilisateur de sélectionner le port série souhaité dans une liste de ports disponibles. Pour plus d'informations, consultez Comment : afficher les ports série disponibles dans Visual Basic.

Cet exemple utilise un bloc **Using** pour vérifier que l'application ferme le port même si une exception est levée. Pour plus d'informations, consultez Using, instruction (Visual Basic).

Dans cet exemple, l'application déconnecte le port série après avoir utilisé le modem. Dans la pratique, vous souhaitez éventuellement transférer des données au modem et vice versa. Pour plus d'informations, consultez Comment : recevoir des chaînes des ports série dans Visual Basic.

Voir aussi

Tâches

Comment : envoyer des chaînes aux ports série dans Visual Basic
Comment : recevoir des chaînes des ports série dans Visual Basic
Comment : afficher les ports série disponibles dans Visual Basic

Référence

My.Computer.Ports, objet
System.IO.Ports.SerialPort

Comment : envoyer des chaînes aux ports série dans Visual Basic

Cette rubrique décrit comment utiliser **My.Computer.Ports** pour envoyer des chaînes aux ports série de l'ordinateur en Visual Basic.

Exemple

Cet exemple envoie une chaîne au port série COM1. Vous pouvez être amené à utiliser un port série différent sur votre ordinateur.

Utilisez la méthode **My.Computer.Ports.OpenSerialPort** pour obtenir une référence au port. Pour plus d'informations, consultez My.Computer.Ports.OpenSerialPort, méthode.

Le bloc **Using** permet à l'application de fermer le port série même s'il génère une exception. Tout le code qui manipule le port série doit apparaître dans ce bloc ou dans un bloc **Try...Catch...Finally**.

La méthode WriteLine envoie les données au port série.

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

Compilation du code

- Cet exemple suppose que l'ordinateur utilise `COM1`.

Programmation fiable

Cet exemple suppose que l'ordinateur utilise `COM1`. Pour une plus grande souplesse, le code doit autoriser l'utilisateur à sélectionner le port souhaité dans une liste de ports disponibles. Pour plus d'informations, consultez Comment : afficher les ports série disponibles dans Visual Basic.

Cet exemple utilise un bloc **Using** pour vérifier que l'application ferme le port même si une exception est levée. Pour plus d'informations, consultez Using, instruction (Visual Basic).

Voir aussi

Tâches

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Comment : afficher les ports série disponibles dans Visual Basic

Référence

My.Computer.Ports, objet
System.IO.Ports.SerialPort

Comment : recevoir des chaînes des ports série dans Visual Basic

Cette rubrique décrit comment utiliser **My.Computer.Ports** pour recevoir des chaînes des ports série de l'ordinateur en Visual Basic.

Pour recevoir des chaînes du port série

1. Initialisez la chaîne de retour.

   VB

   ```
   Dim returnStr As String = ""
   ```

2. Déterminez quel port série doit fournir les chaînes. Cet exemple suppose que c'est le port `COM1`.

3. Utilisez la méthode **My.Computer.Ports.OpenSerialPort** pour obtenir une référence au port. Pour plus d'informations, consultez My.Computer.Ports.OpenSerialPort, méthode.

   Le bloc **Using** permet à l'application de fermer le port série même s'il génère une exception. Tout le code qui manipule le port série doit apparaître dans ce bloc ou dans un bloc **Try...Catch...Finally**.

   VB

   ```
   Using com1 As IO.Ports.SerialPort = _
           My.Computer.Ports.OpenSerialPort("COM1")
   End Using
   ```

4. Créez une boucle **Do** pour lire les lignes de texte jusqu'à ce qu'il ne reste plus aucune ligne disponible.

   VB

   ```
   Do
   Loop
   ```

5. Utilisez la méthode ReadLine pour lire la ligne de texte disponible suivante à partir du port série.

   VB

   ```
   Dim Incoming As String = com1.ReadLine()
   ```

6. Utilisez une instruction **If** pour déterminer si la méthode **ReadLine** retourne **Nothing** (ce qui signifie qu'il n'y a plus de texte disponible). Si elle retourne **Nothing**, quittez la boucle **Do**.

   VB

   ```
   If Incoming Is Nothing Then
       Exit Do
   End If
   ```

7. Ajoutez un bloc **Else** à l'instruction **If** pour gérer le cas si la chaîne est effectivement lue. Le bloc ajoute la chaîne du port série à la chaîne de retour.

   VB

   ```
   Else
   ```

```
        returnStr &= Incoming & vbCrLf
```

8. Retournez la chaîne.

   VB

```
    Return returnStr
```

Exemple

VB

```
Function ReceiveSerialData() As String
    ' Receive strings from a serial port.
    Dim returnStr As String = ""

    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        Do
            Dim Incoming As String = com1.ReadLine()
            If Incoming Is Nothing Then
                Exit Do
            Else
                returnStr &= Incoming & vbCrLf
            End If
        Loop
    End Using

    Return returnStr
End Function
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Connectivité et réseau**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple suppose que l'ordinateur utilise `COM1`.

Programmation fiable

Cet exemple suppose que l'ordinateur utilise `COM1`. Pour plus de souplesse, le code doit permettre à l'utilisateur de sélectionner le port série dans une liste de ports disponibles. Pour plus d'informations, consultez Comment : afficher les ports série disponibles dans Visual Basic.

Cet exemple utilise un bloc **Using** pour vérifier que l'application ferme le port même si une exception est levée. Pour plus d'informations, consultez Using, instruction (Visual Basic).

Voir aussi

Tâches

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Comment : envoyer des chaînes aux ports série dans Visual Basic
Comment : afficher les ports série disponibles dans Visual Basic

Référence

My.Computer.Ports, objet
System.IO.Ports.SerialPort

Lecture et écriture dans le Registre

Répertorie les rubriques conceptuelles et les rubriques de tâches associées au Registre.

Lorsque vous effectuez de la programmation dans Visual Basic 2005, vous pouvez choisir d'accéder au Registre soit via les fonctions fournies par Visual Basic 2005, soit via les classes de Registre du .NET Framework. Le Registre contient des informations provenant du système d'exploitation et des applications hébergées sur l'ordinateur. L'utilisation du Registre peut compromettre la sécurité par l'accord d'accès inappropriés aux ressources du système ou aux informations protégées.

Dans cette section

Tâches courantes relatives au Registre

Répertorie les tâches associées au Registre, par exemple déterminer ou définir la valeur d'une clé de Registre.

Lecture et écriture dans le Registre à l'aide de My

Explique comment utiliser l'objet **My.Computer.Registry** pour accéder au Registre.

Lecture et écriture dans le Registre à l'aide de l'espace de noms Microsoft.Win32

Explique comment utiliser les fonctions de Visual Basic 2005, **DeleteSetting**, **GetSetting**, **GetAllSettings** et **SaveSetting** pour accéder au Registre.

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Explique comment créer une clé de Registre et ses sous-clés, comment modifier leurs paramètres et supprimer des clés.

Sécurité et Registre

Décrit les problèmes de sécurité liés à l'utilisation du Registre.

Dépannage : manipulation du Registre

Répertorie les problèmes couramment rencontrés lors de l'accès au Registre et explique comment les résoudre.

Rubriques connexes

My.Computer.Registry, objet

Répertorie et décrit les membres de l'objet **My.Computer.Registry**.

Registry, classe

Fournit une vue d'ensemble de la classe **Registry**, avec des liens vers les clés et les membres.

Tâches courantes relatives au Registre

L'objet **My.Computer.Registry** fournit des propriétés et des méthodes pour accéder au Registre.

Tâches

Le tableau suivant répertorie des exemples qui concernent le Registre.

Pour	Reportez-vous à
Créer une clé de Registre et définir ses valeurs	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre spécifique	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

Concepts

Lecture et écriture dans le Registre à l'aide de My

Sécurité et Registre

Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic

La méthode **CreateSubKey** de l'objet **My.Computer.Registry** peut être utilisée pour créer une clé de Registre.

Procédure

Pour créer une clé de Registre

- Utilisez la méthode **CreateSubKey**, en spécifiant sous quelle ruche placer la clé ainsi que le nom de la clé. Le paramètre `Subkey` ne respecte pas la casse. Cet exemple crée la clé de Registre `MyTestKey` sous HKEY_CURRENT_USER.

  VB

  ```
  My.Computer.Registry.CurrentUser.CreateSubKey("MyTestKey")
  ```

Pour créer une clé de Registre et y définir une valeur

1. Utilisez la méthode **CreateSubkey**, en spécifiant sous quelle ruche placer la clé ainsi que le nom de la clé. Cet exemple crée la clé de Registre `MyTestKey` sous HKEY_CURRENT_USER.

   VB

   ```
   My.Computer.Registry.CurrentUser.CreateSubKey("MyTestKey")
   ```

2. Définissez la valeur avec la méthode **SetValue**. Cet exemple définit la valeur de chaîne "MyTestKeyValue" sur "This is a test value".

   VB

   ```
   My.Computer.Registry.SetValue("HKEY_CURRENT_USER\MyTestKey", _
   "MyTestKeyValue", "This is a test value.")
   ```

Exemple

Cet exemple crée la clé de Registre `MyTestKey` sous HKEY_CURRENT_USER, puis définit la valeur de chaîne `MyTestKeyValue` à `This is a test value`.

VB

```
My.Computer.Registry.CurrentUser.CreateSubKey("MyTestKey")
' Change MyTestKeyValue to This is a test value.
My.Computer.Registry.SetValue("HKEY_CURRENT_USER\MyTestKey", _
"MyTestKeyValue", "This is a test value.")
```

Programmation fiable

Examinez la structure du Registre afin de trouver un emplacement approprié pour votre clé. Par exemple, vous souhaiterez peut-être ouvrir la clé HKEY_CURRENT_USER\Software de l'utilisateur actuel et créer une clé avec le nom de votre société. Vous pouvez ensuite ajouter les valeurs du Registre à la clé de votre société.

Lors de la lecture du Registre à partir d'une application Web, l'utilisateur en cours dépend de l'authentification et de l'emprunt d'identité implémentés dans l'application Web.

Pour des raisons de sécurité, il est préférable d'écrire des données dans le dossier utilisateur (CurrentUser) plutôt que sur l'ordinateur local (LocalMachine).

Lorsque vous créez une valeur de Registre, vous devez déterminer la marche à suivre si cette valeur existe déjà. Il est possible

qu'un autre processus, peut-être nuisible, ait déjà créé la valeur et puisse y accéder. Lorsque vous placez des données dans la valeur du Registre, elles sont disponibles pour les autres processus. Pour empêcher cela, utilisez la méthode GetValue. Elle retourne **Nothing** si la clé n'existe pas.

Pour des raisons de sécurité, il est déconseillé de stocker des données confidentielles, telles que des mots de passe, en texte brut dans le Registre, même si la clé de Registre est protégée par des listes de contrôle d'accès (ACL, Access Control Lists).

Les conditions ci-dessous peuvent générer une exception.

- Le nom de la clé est **Nothing** (ArgumentNullException).
- L'utilisateur n'a pas l'autorisation de créer des clés de Registre (SecurityException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).
- La clé est fermée (IOException).
- La clé de Registre est en lecture seule (UnauthorizedAccessException).

Sécurité

Pour exécuter ce processus, votre assembly requiert un niveau de privilège accordé par la classe RegistryPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le processus peut lever une exception en raison de privilèges insuffisants. De même, l'utilisateur doit posséder les listes de contrôles d'accès appropriées pour pouvoir créer des paramètres ou écrire dans ceux-ci. Par exemple, une application locale disposant d'une autorisation de sécurité d'accès du code peut ne pas avoir une autorisation du système d'exploitation. Pour plus d'informations, consultez
Notions fondamentales de la sécurité d'accès du code.

Voir aussi

Tâches

Dépannage : manipulation du Registre
Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

My.Computer.Registry, objet
My.Computer.Registry.CurrentUser, propriété
CreateSubKey

Concepts

Tâches courantes relatives au Registre
Notions fondamentales de la sécurité d'accès du code

Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic

La méthode **GetValue** de l'objet **My.Computer.Registry** peut être utilisée pour déterminer si une valeur donnée existe dans une clé de Registre spécifique.

Lors de la lecture du Registre à partir d'une application Web, l'utilisateur en cours dépend de l'authentification et de l'emprunt d'identité implémentés dans l'application Web.

Pour déterminer si une valeur existe dans une clé de Registre

- Utilisez la méthode **GetValue** pour récupérer la valeur. Le code suivant vérifie la valeur et retourne un message si elle n'existe pas.

  VB

```
If My.Computer.Registry.GetValue("HKEY_LOCAL_MACHINE\TestApp", _
"TestValue", Nothing) Is Nothing Then
  MsgBox("Value does not exist.")
End If
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception.

- Le nom de la clé est **Nothing** (ArgumentNullException).
- L'utilisateur n'a pas l'autorisation de créer des clés de Registre (SecurityException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).
- La clé est fermée (IOException).
- La clé de Registre est en lecture seule (UnauthorizedAccessException).

Sécurité

Pour exécuter ce processus, votre assembly requiert un niveau de privilège accordé par la classe RegistryPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le processus peut lever une exception en raison de privilèges insuffisants. De même, l'utilisateur doit posséder les listes de contrôle d'accès appropriées pour pouvoir créer des paramètres ou écrire dans ceux-ci. Par exemple, une application locale disposant d'une autorisation de sécurité d'accès du code peut ne pas avoir une autorisation du système d'exploitation. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

My.Computer.Registry, objet

My.Computer.Registry.CurrentUser, propriété

Concepts

Notions fondamentales de la sécurité d'accès du code

Tâches courantes relatives au Registre

Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic

La méthode **GetValue** de l'objet **My.Computer.Registry** peut être utilisée pour lire des valeurs dans le Registre de Windows. Si la clé, dans ce cas "Software\MyApp", n'existe pas, une exception est levée. Si *ValueName* (dans ce cas-ci « Name ») n'existe pas, **Nothing** est retourné.

Pour lire une valeur à partir d'une clé de Registre

- Utilisez la méthode **GetValue** (en spécifiant le chemin d'accès et le nom) pour lire une valeur de clé de Registre. L'exemple suivant lit la valeur `Name` de `HKEY_CURRENT_USER\Software\MyApp` et l'affiche dans une boîte de message.

  VB

```
Dim readValue As String
readValue = My.Computer.Registry.GetValue _
("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
MsgBox("The value is " & readValue)
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Système d'exploitation Windows > Registre**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Le Registre contient des clés de niveau supérieur (ou racine) qui sont utilisées pour stocker des données. Par exemple, la clé racine HKEY_LOCAL_MACHINE est utilisée pour stocker des paramètres au niveau de l'ordinateur utilisés par tous les utilisateurs, tandis que HKEY_CURRENT_USER est utilisé pour stocker des données spécifiques à un utilisateur individuel.

Les conditions ci-dessous peuvent générer une exception.

- Le nom de la clé est **Nothing** (ArgumentNullException).
- L'utilisateur n'a pas l'autorisation de lire à partir de clés de Registre (SecurityException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).

Sécurité

Pour exécuter ce processus, votre assembly requiert un niveau de privilège accordé par la classe RegistryPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le processus peut lever une exception en raison de privilèges insuffisants. De même, l'utilisateur doit posséder les listes de contrôles d'accès appropriées pour pouvoir créer des paramètres ou écrire dans ceux-ci. Par exemple, une application locale disposant d'une autorisation de sécurité d'accès du code peut ne pas avoir une autorisation du système d'exploitation. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

RegistryHive

Concepts

Tâches courantes relatives au Registre

Comment : définir les valeurs dans les clés de Registre dans Visual Basic

La méthode **SetValue** de l'objet **My.Computer.Registry** peut être utilisée pour écrire des valeurs dans le Registre de Windows. Le Registre contient des clés de niveau supérieur (ou racine) qui sont utilisées pour stocker des données. Par exemple, la clé racine HKEY_LOCAL_MACHINE est utilisée pour stocker des paramètres au niveau de l'ordinateur utilisés par tous les utilisateurs, tandis que HKEY_CURRENT_USER est utilisé pour stocker des données spécifiques à un utilisateur individuel.

La valeur, y compris le chemin d'accès complet à la clé, est créée si elle n'existe pas.

Procédure

Pour écrire une valeur dans une clé de Registre

- Utilisez la méthode **SetValue**, en spécifiant la clé et la valeur. Cet exemple définit la valeur `Name` à "Author's Name" dans la clé HKEY_CURRENT_USER\Software\TestApp.

  VB

  ```
  My.Computer.Registry.SetValue _
  ("HKEY_CURRENT_USER\Software\TestApp", "Name", "Author's Name")
  ```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Système d'exploitation Windows > Registre**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Les données de préférence de l'utilisateur doivent être écrites dans la ruche `Microsoft.Win32.Registry.CurrentUser`.

Pour des raisons de sécurité, il est déconseillé de stocker des données confidentielles, telles que des mots de passe, en texte brut dans le Registre, même si la clé de Registre est protégée par des listes de contrôle d'accès (ACL, Access Control Lists).

Les conditions ci-dessous peuvent générer une exception.

- Le nom de la clé est **Nothing** (ArgumentNullException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).
- La ruche spécifiée n'est pas valide (**ArgumentException**).
- La clé est fermée (IOException).
- Le chemin d'accès n'est pas valide (**IOException**).
- La clé de Registre est en lecture seule (UnauthorizedAccessException).

Sécurité

Pour exécuter ce processus, votre assembly requiert un niveau de privilège accordé par la classe RegistryPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le processus peut lever une exception en raison de privilèges insuffisants. De même, l'utilisateur doit posséder les listes de contrôles d'accès appropriées pour pouvoir créer des paramètres ou écrire dans ceux-ci. Par exemple, une application locale disposant d'une autorisation de sécurité d'accès du code peut ne pas avoir une autorisation du système d'exploitation. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Voir aussi

Tâches

Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs
Dépannage : manipulation du Registre

Référence

My.Computer.Registry, membres de l'objet

My.Computer.Registry.SetValue, méthode

Concepts

Tâches courantes relatives au Registre

Comment : supprimer une clé de Registre dans Visual Basic

Les méthodes DeleteSubKey et DeleteSubKey peuvent être utilisées pour supprimer des clés de Registre.

Procédure

Pour supprimer une clé de Registre

- Utilisez la méthode `DeleteSubKey` pour supprimer une clé de Registre. Cet exemple supprime la clé Software/TestApp dans la ruche CurrentUser. Vous pouvez spécifier la chaîne appropriée dans le code ou faire en sorte que celui-ci repose sur des informations fournies par l'utilisateur.

  VB

```
My.Computer.Registry.CurrentUser.DeleteSubKey(text)
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Système d'exploitation Windows > Registre**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

La méthode `DeleteSubKey` retourne une chaîne vide si la paire clé/valeur n'existe pas.

Les conditions ci-dessous peuvent générer une exception.

- Le nom de la clé est **Nothing** (ArgumentNullException).
- L'utilisateur n'a pas l'autorisation de supprimer des clés de Registre (SecurityException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).
- La clé de Registre est en lecture seule (UnauthorizedAccessException).

Sécurité

Les appels au Registre échouent lorsque l'utilisateur ne dispose pas des autorisations d'exécution nécessaires (RegistryPermission) ou de l'accès correct (tel que déterminé par les listes de contrôle d'accès) pour créer ou modifier des paramètres. Par exemple, une application locale disposant d'une autorisation de sécurité d'accès du code peut ne pas avoir une autorisation du système d'exploitation.

Voir aussi

Tâches

Dépannage : manipulation du Registre
Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

DeleteSubKey
DeleteSubKey
RegistryKey

Concepts

Sécurité et Registre
Tâches courantes relatives au Registre

Lecture et écriture dans le Registre à l'aide de My

L'objet **My.Computer.Registry** offre des méthodes et des propriétés permettant d'utiliser les clés de Registre.

Le Registre Windows contient des informations provenant du système d'exploitation et des applications hébergées sur l'ordinateur. Bien que **My.Settings** soit le paramètre recommandé pour stocker ce type d'informations, les applications héritées (legacy) peuvent faire appel au Registre.

Pour une liste complète des membres **My.Computer.Registry**, consultez My.Computer.Registry, membres de l'objet.

Tâches

Le tableau suivant répertorie des exemples de tâches pour des clés de Registre concernant l'objet **My.Computer.Registry**.

Pour	Reportez-vous à
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic

Sécurité et Registre

L'utilisation du Registre peut compromettre la sécurité par l'accord d'accès inappropriés aux ressources du système ou aux informations protégées. Pour utiliser ces propriétés, vous devez disposer d'autorisations en lecture et en écriture à partir de l'énumération RegistryPermissionAccess qui contrôle l'accès aux variables de Registre. Tout code s'exécutant avec une confiance totale (dans la stratégie de sécurité par défaut, tout code installé sur le disque dur local de l'utilisateur) détient les autorisations nécessaires pour accéder au Registre. Pour plus d'informations, consultez T:System.Security.Permissions.RegistryPermission.

Les variables de Registre ne doivent pas être stockées dans des emplacements de la mémoire auxquels du code sans RegistryPermission peut accéder. De même, lorsque vous accordez des autorisations, accordez les privilèges minimaux nécessaires à l'accomplissement des tâches.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

My.Settings, objet

Concepts

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de l'espace de noms Microsoft.Win32

Même si **My.Computer.Registry** doit couvrir vos besoins de base lorsque vous programmez le Registre, vous pouvez utiliser également les classes Registry et RegistryKey dans l'espace de noms Microsoft.Win32 du .NET Framework.

Clés dans la classe de Registre

La classe **Registry** fournit les clés de Registre de base qui permettent d'accéder aux sous-clés et aux valeurs associées. Les clés de base sont en lecture seule. Le tableau ci-après répertorie et décrit les sept clés exposées par la classe **Registry**.

Clé	Description
ClassesRoot	Définit les types de documents et les propriétés associées.
CurrentConfig	Contient des informations sur la configuration matérielle qui ne sont pas spécifiques à un utilisateur.
CurrentUser	Contient des informations sur les préférences de l'utilisateur en cours (par exemple, des variables d'environnement).
DynData	Contient des données de Registre dynamiques (par exemple, les données utilisées par les pilotes de périphériques virtuels).
LocalMachine	Contient cinq sous-clés (Hardware, SAM, Security, Software et System) qui stockent les données sur la configuration de l'ordinateur local.
PerformanceData	Contient des informations sur les performances des composants logiciels.
Utilisateurs	Contient des informations sur les préférences par défaut des utilisateurs.

Remarque de sécurité

Pour des raisons de sécurité, il est préférable d'écrire des données dans l'utilisateur en cours (CurrentUser) que sur l'ordinateur local (LocalMachine). Il existe un cas, généralement appelé « squattage », où la clé que vous créez a déjà été créée par un autre processus, éventuellement malveillant. Pour empêcher cette situation, utilisez une méthode, telle que GetValue qui retourne **Nothing** si la clé n'existe pas déjà.

Lecture d'une valeur du Registre

Le code suivant montre comment lire une chaîne à partir de HKEY_CURRENT_USER.

VB

```
Dim regVersion As Microsoft.Win32.RegistryKey
Dim keyValue As String
keyValue = "Software\\Microsoft\\TestApp\\1.0"
regVersion = Microsoft.Win32.Registry.CurrentUser.OpenSubKey(keyValue, False)
Dim intVersion As Integer = 0
If (Not regVersion Is Nothing) Then
    intVersion = regVersion.GetValue("Version", 0)
    regVersion.Close()
End If
```

Le code suivant lit, incrémente, puis écrit une chaîne dans HKEY_CURRENT_USER.

VB

```
Dim regVersion As Microsoft.Win32.RegistryKey
regVersion = Microsoft.Win32.Registry.CurrentUser.OpenSubKey( _
```

```
            "SOFTWARE\\Microsoft\\TestApp\\1.0", True)
If regVersion Is Nothing Then
    ' Key doesn't exist; create it.
    regVersion = Microsoft.Win32.Registry.CurrentUser.CreateSubKey( _
                 "SOFTWARE\\Microsoft\\TestApp\\1.0")
End If

Dim intVersion As Integer = 0
If (Not regVersion Is Nothing) Then
    intVersion = regVersion.GetValue("Version", 0)
    intVersion = intVersion + 1
    regVersion.SetValue("Version", intVersion)
    regVersion.Close()
End If
```

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

SystemException Class

ApplicationException Class

My.Computer.Registry, objet

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Cette procédure pas à pas explique comment créer une application qui accédera à des clés de Registre sur l'ordinateur afin que les utilisateurs puissent créer et supprimer des clés, et montre comment lire, obtenir, définir et supprimer des valeurs.

Pour créer le formulaire principal

1. Sélectionnez **Nouveau projet** dans le menu **Fichier** et cliquez sur **Application Windows**.
2. Ajoutez un TextBox nommé `Value` au formulaire. Dans la fenêtre **Propriétés** dans le coin inférieur droit, dans le champ **(Nom)**, tapez **Valeur**.
3. Ajoutez un ListBox nommé `History` au formulaire. Dans la fenêtre **Propriétés** dans le coin inférieur droit, dans le champ **(Nom)**, tapez **Historique**.
4. Créez la variable supplémentaire et ajoutez-la juste après la déclaration de classe.

   VB

   ```
   Dim tempKey As Microsoft.Win32.RegistryKey
   ```

Pour parcourir des clés de Registre dans une zone de liste déroulante

1. Ajoutez à votre formulaire un ComboBox nommé `selectHive`, qui affichera les ruches de Registre et vous permettra d'en sélectionner une. Remplissez-le en ajoutant le code suivant à l'événement de chargement du formulaire.

   VB

   ```
   selectHive.Items.Add("ClassesRoot")
   selectHive.Items.Add("CurentConfig")
   selectHive.Items.Add("CurrentUser")
   selectHive.Items.Add("DynData")
   selectHive.Items.Add("LocalMachine")
   selectHive.Items.Add("PerformanceData")
   selectHive.Items.Add("Users")
   ```

2. Insérez le code suivant après votre déclaration de classe.

   VB

   ```
   Dim registryObject As Microsoft.Win32.RegistryKey = Nothing
   ```

3. Ajoutez le code suivant à l'événement SelectedIndexChanged de `selectHive`.

   VB

   ```
   Select Case selectHive.Text
     Case "ClassesRoot"
       registryObject = My.Computer.Registry.ClassesRoot
     Case "CurrentConfig"
       registryObject = My.Computer.Registry.CurrentConfig
     Case "CurrentUser"
       registryObject = My.Computer.Registry.CurrentUser
     Case "DynData"
       registryObject = My.Computer.Registry.DynData
     Case "LocalMachine"
   ```

```
    registryObject = My.Computer.Registry.LocalMachine
  Case "PerformanceData"
    registryObject = My.Computer.Registry.PerformanceData
  Case "Users"
    registryObject = My.Computer.Registry.Users
End Select
```

Pour lire une valeur dans une clé de Registre

1. Ajoutez au formulaire un Button nommé `ReadValueButton` avec le texte « Lecture de valeur ».
2. Ajoutez au formulaire un **TextBox** nommé `showSubKey` avec le texte « Saisie de sous-clé ».
3. Ajoutez le code suivant à l'événement Click de `ReadValueButton`.

   VB
   ```
   tempKey = registryObject
   If tempKey Is Nothing Then
     MsgBox("Please select a registry hive.")
     Return
   End If
   Value.Text = CStr(tempKey.GetValue(ShowSubKey.Text))
   History.Items.Add("Read Value " & selectHive.Text & _
                     "\" & ShowSubKey.Text)
   ```
4. Testez votre application en introduisant le nom d'une sous-clé existante dans la zone de texte `showSubKey`. Lorsque vous cliquez sur le `ReadValueButton`, la zone de texte `Value` affiche la valeur.

Pour définir une valeur dans une clé de Registre

1. Ajoutez au formulaire un bouton nommé `SetValueButton` avec le texte « Définition de valeur ».
2. Ajoutez le code suivant à son événement `Click`.

   VB
   ```
   tempKey = registryObject
   If tempKey Is Nothing Then
     MsgBox("Please select a registry hive.")
     Return
   End If
   If Value.Text Is Nothing Then
     MsgBox("Please enter a value.")
     Return
   End If
   tempKey.SetValue(showSubKey.Text, Value.Text)
   tempKey.Close()
   History.Items.Add("Set Value " & selectHive.Text & _
                     "\" & showSubKey.Text)
   ```
3. Testez votre application en entrant une nouvelle valeur pour une sous-clé dans la zone de texte **Valeur**, puis en confirmant que la valeur a été modifiée avec le bouton nommé `ReadValueButton`.

Pour créer une clé de Registre

1. Ajoutez au formulaire un bouton nommé `CreateButton` avec comme texte « Création de clé ».

2. Ajoutez le code suivant à son événement **Click**.

   VB

```
registryObject.CreateSubKey(showSubKey.Text)
History.Items.Add("Create Key " & selectHive.Text & _
                  "\" & showSubKey.Text)
```

3. Testez votre application en entrant un nouveau nom de clé dans la zone de texte **showSubKey** et en utilisant l'**Éditeur du Registre** pour confirmer que votre clé a été créée.

Pour supprimer une clé de Registre

1. Ajoutez au formulaire un bouton nommé `DeleteButton` avec comme texte « Suppression de clé ».

2. Ajoutez le code suivant à son événement **Click**.

   VB

```
tempKey = registryObject
If tempKey Is Nothing Then
  MsgBox("Please select a registry hive.")
  Return
End If
If showSubKey.Text Is Nothing Then
  MsgBox("Please enter a subkey.")
  Return
End If
registryObject.DeleteSubKey(showSubKey.Text)
History.Items.Add("Delete Key " & selectHive.Text & _
                  "\" & showSubKey.Text)
```

3. Testez votre code en supprimant une sous-clé et en utilisant l' **Éditeur du Registre** pour confirmer que la clé a été supprimée.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Concepts

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

Lecture et écriture dans le Registre à l'aide de l'espace de noms Microsoft.Win32

Sécurité et Registre

Cette page discute des conséquences pour la sécurité liées au stockage de données dans le Registre.

Autorisations

Pour des raisons de sécurité, il est déconseillé de stocker des données confidentielles, telles que des mots de passe, en texte brut dans le Registre, même si la clé de Registre est protégée par des listes de contrôle d'accès (ACL, Access Control Lists).

L'utilisation du Registre peut compromettre la sécurité par l'accord d'accès inappropriés aux ressources du système ou aux informations protégées. Pour utiliser ces propriétés, vous devez posséder des autorisations de lecture et d'écriture de l'énumération RegistryPermissionAccess, qui contrôle l'accès aux variables de Registre. Tout code s'exécutant avec une confiance totale (dans la stratégie de sécurité par défaut, tout code installé sur le disque dur local de l'utilisateur) détient les autorisations nécessaires pour accéder au Registre. Pour plus d'informations, consultez T:System.Security.Permissions.RegistryPermission.

Les variables de Registre ne doivent pas être stockées dans des emplacements mémoire où du code sans RegistryPermission peut y accéder. De même, lorsque vous accordez des autorisations, accordez les privilèges minimaux nécessaires pour effectuer le travail.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération **RegistryPermissionAccess**. Le tableau suivant décrit en détail ses membres.

Valeur	Accès aux variables de Registre
AllAccess	Création, lecture et écriture
Create	Créer
NoAccess	Pas d'accès
Read	Lecture
Write	Écriture

Vérification des valeurs dans les clés de Registre

Lorsque vous créez une valeur de Registre, vous devez déterminer la marche à suivre si cette valeur existe déjà. Il est possible qu'un autre processus, peut-être nuisible, ait déjà créé la valeur et puisse y accéder. Lorsque vous placez des données dans la valeur du Registre, elles sont disponibles pour les autres processus. Pour empêcher cela, utilisez la méthode **GetValue**. Elle retourne **Nothing** si la clé n'existe pas.

Remarque de sécurité
Lors de la lecture du Registre à partir d'une application Web, l'identité de l'utilisateur actuel dépend de l'authentification et de l'emprunt d'identité implémentés dans l'application Web.

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

My.Computer.Registry, membres de l'objet

Concepts

Tâches courantes relatives au Registre

Lecture et écriture dans le Registre à l'aide de My

Dépannage : manipulation du Registre

Lorsque vous utilisez le Registre, vous pouvez rencontrer des exceptions spécifiques. Cette rubrique répertorie les exceptions les plus communes et propose des solutions pour les résoudre.

Exceptions

Le tableau suivant répertorie les exceptions qui peuvent se produire lorsque vous utilisez le Registre et fournit des conseils sur la façon de les résoudre.

Exception	Suggestion
ArgumentException	Assurez-vous que la clé sur laquelle vous exécutez une opération existe, et que le chemin d'accès que vous avez fourni est correct et ne comporte pas plus de 255 caractères.
SecurityException	Assurez-vous que vous possédez les autorisations de lecture et d'écriture dans RegistryPermissionAccess.
UnauthorizedAccessException	Assurez-vous que la clé a été ouverte en écriture si vous voulez écrire dans cette clé.

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

My.Computer.Registry, objet

Concepts

Tâches courantes relatives au Registre
Lecture et écriture dans le Registre à l'aide de My
Lecture et écriture dans le Registre à l'aide de l'espace de noms Microsoft.Win32
Sécurité et Registre

Accès à d'autres processus

Cette page répertorie des rubriques qui expliquent comment accéder à d'autres processus à l'aide de la classe Process.

Dans cette section

Comment : arrêter une application (Visual Basic)

Explique comment arrêter une application en cours d'exécution.

Comment : afficher les processus (Visual Basic)

Explique comment obtenir une liste des processus d'applications en cours d'exécution.

Rubriques connexes

Comment : démarrer une application et envoyer des séquences de touches (Visual Basic)

Explique comment démarrer une application.

Shell, fonction

Exécute un programme exécutable et retourne un entier contenant l'ID du processus du programme s'il est toujours en cours d'exécution.

Comment : arrêter une application (Visual Basic)

Cet exemple arrête une application en cours d'exécution.

Exemple

VB

```
Dim pList() As System.Diagnostics.Process = _
    System.Diagnostics.Process.GetProcessesByName("notepad")
For Each proc As System.Diagnostics.Process In pList
    Dim resp As MsgBoxResult
    resp = MsgBox("Terminate " & proc.ProcessName & "?", _
        MsgBoxStyle.YesNo, "Terminate?")
    If resp = MsgBoxResult.Yes Then
        proc.Kill()
    End If
Next
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Système d'exploitation Windows > Processus**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite :

- Une instruction **Imports** spécifiant l'espace de noms System.Diagnostics. Pour plus d'informations, consultez Imports, instruction.
- Une instance en cours d'exécution de l'application Bloc-notes (Notepad.exe).

Sécurité

Dans la mesure où cet exemple met immédiatement fin aux processus, il peut provoquer des pertes de données lorsqu'il est utilisé avec certaines applications.

La méthode Kill requiert une confiance totale (classe SecurityException).

Voir aussi

Référence

Process

Autres ressources

Accès à d'autres processus

Comment : afficher les processus (Visual Basic)

Cet exemple crée une liste des processus d'applications en cours d'exécution. Ces informations peuvent être utilisées lors de l'appel de la fonction **AppActivate**.

Exemple

VB

```
Dim pList() As System.Diagnostics.Process = _
    System.Diagnostics.Process.GetProcesses
For Each proc As System.Diagnostics.Process In pList
    MsgBox(proc.ProcessName)
Next
```

Compilation du code

Cet exemple nécessite :

- Une instruction **Imports** spécifiant l'espace de noms System.Diagnostics. Pour plus d'informations, consultez Imports, instruction.

Voir aussi

Référence

AppActivate, fonction

Process

Autres ressources

Exemples de Windows Forms Visual Basic

Mise à jour d'une application

La propriété **My.Application.Deployment** peut être utilisée pour rechercher les mises à jour disponibles pour l'application. Si l'application a été déployée comme une application ClickOnce, l'exemple affiche le texte d'interface utilisateur approprié.

Dans cette section

Ce tableau répertorie les tâches associées à la propriété **My.Application.Deployment** et indique les rubriques qui expliquent comment accomplir chaque tâche.

Pour	Reportez-vous à
Rechercher les mises à jour de l'application	Comment : rechercher les mises à jour de l'application ClickOnce
Télécharger les mises à jour de l'application	Comment : télécharger les mises à jour d'applications ClickOnce

Voir aussi

Référence

My.Application.Deployment, propriété
My.Application, objet

Comment : rechercher les mises à jour de l'application ClickOnce

Cet exemple utilise l'objet **My.Application.Deployment** pour rechercher les mises à jour disponibles pour l'application. Si l'application a été déployée comme une application ClickOnce, l'exemple affiche le texte d'interface utilisateur approprié.

Pour plus d'informations sur les applications ClickOnce et comment les déployer, consultez Déploiement ClickOnce et Publication d'applications ClickOnce.

Exemple

Cet exemple utilise la propriété My.Application.IsNetworkDeployed pour vérifier que l'application est déployée à partir d'un réseau à l'aide de ClickOnce. Ensuite, il utilise la méthode CheckForUpdate de l'objet **My.Application.Deployment** pour vérifier si une mise à jour ClickOnce est disponible pour l'application.

VB

```
Sub CheckUpdateAvailability()
    If My.Application.IsNetworkDeployed() Then
        If My.Application.Deployment.CheckForUpdate() Then
            MsgBox("Update is available for download")
        Else
            MsgBox("No updates are available for download")
        End If
    Else
        MsgBox("Application is not ClickOnce deployed")
    End If
End Sub
```

Seules les applications qui sont déployées via ClickOnce peuvent être mises à jour à l'aide de l'objet **My.Application.Deployment**. Pour plus d'informations sur le déploiement d'une application ClickOnce, consultez Comment : publier une application ClickOnce.

Voir aussi

Tâches

Comment : télécharger les mises à jour d'applications ClickOnce

Référence

My.Application.Deployment, propriété

Comment : télécharger les mises à jour d'applications ClickOnce

Cet exemple utilise l'objet **My.Application.Deployment** pour télécharger et installer la version la plus récente d'une application. L'exemple ne met pas à jour l'application à moins qu'elle ne soit obsolète et ait été déployée comme une application ClickOnce.

Pour plus d'informations sur les applications ClickOnce et comment les déployer, consultez Déploiement ClickOnce et Publication d'applications ClickOnce.

Exemple

Cet exemple télécharge et installe la mise à jour après avoir utilisé la propriété My.Application.IsNetworkDeployed pour vérifier que l'application est déployée à l'aide de ClickOnce. La méthode Update ne met pas à jour l'application tant qu'elle n'est pas obsolète. Pour pouvoir utiliser la mise à jour, l'application doit redémarrer.

VB

```
Sub UpdateApplication()
    If My.Application.IsNetworkDeployed Then
        My.Application.Deployment.Update()
    End If
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Application—Compilation, Ressources et Paramètres**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Seules les applications qui sont déployées via ClickOnce peuvent être mises à jour à l'aide de l'objet **My.Application.Deployment**. Pour plus d'informations sur le déploiement d'une application ClickOnce, consultez Comment : publier une application ClickOnce.

Voir aussi

Tâches

Comment : rechercher les mises à jour de l'application ClickOnce

Référence

My.Application.Deployment, propriété

Enregistrement d'informations provenant de l'application

Cette section contient des rubriques qui expliquent comment enregistrer les informations provenant de votre application à l'aide de l'objet **My.Application.Log** ou **My.Log**, et comment étendre les fonctions de journalisation de l'application.

L'objet **Log** fournit des méthodes permettant d'écrire les informations dans les écouteurs de journalisation de l'application, tandis que la propriété **TraceSource** avancée de l'objet **Log** fournit des informations de configuration détaillées. L'objet **Log** est configuré par le fichier de configuration de l'application.

L'objet **My.Log** n'est disponible que pour les applications ASP.NET. Pour les applications clientes, utilisez **My.Application.Log**. Pour plus d'informations, consultez My.Application.Log, objet et My.Log, objet.

Tâches

Pour	Reportez-vous à
Écrire les informations sur l'événement dans les journaux de l'application.	Comment : écrire des messages de journal
Écrire les informations sur l'exception dans les journaux de l'application.	Comment : enregistrer des exceptions dans Visual Basic
Écrire les informations de traçage dans les journaux de l'application au démarrage et à l'arrêt de l'application.	Comment : enregistrer des messages lorsque l'application démarre ou s'arrête
Configurer **My.Application.Log** pour écrire les informations dans un fichier texte.	Comment : écrire des informations sur des événements dans un fichier texte
Configurer **My.Application.Log** pour écrire les informations dans un journal des événements.	Comment : écrire dans le journal des événements de l'application
Modifier l'emplacement où **My.Application.Log** écrit les informations.	Procédure pas à pas : modification de l'emplacement des informations My.Application.Log
Déterminer à quel emplacement **My.Application.Log** écrit les informations.	Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log
Créer un écouteur de journalisation personnalisé pour **My.Application.Log**.	Procédure pas à pas : création d'écouteurs de journalisation personnalisés
Filtrer la sortie des journaux **My.Application.Log**.	Procédure pas à pas : filtrage de la sortie de My.Application.Log

Voir aussi

Tâches

Dépannage : écouteurs de journalisation

Référence

My.Application.Log, objet

My.Log, objet

Concepts

Utilisation des journaux d'applications dans Visual Basic
Enregistrement et traçage dans le .NET Framework avec Visual Basic

Comment : écrire des messages de journal

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour enregistrer des informations sur votre application. Cet exemple indique comment utiliser la méthode **My.Application.Log.WriteEntry** pour enregistrer des informations de traçage.

Pour enregistrer des informations sur les exceptions, utilisez la méthode **My.Application.Log.WriteException** ; consultez Comment : enregistrer des exceptions dans Visual Basic.

Exemple

Cet exemple utilise la méthode **My.Application.Log.WriteEntry** pour écrire des informations de traçage.

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

Sécurité

Vérifiez que les données que vous écrivez dans le journal ne comprennent pas des informations sensibles telles que des mots de passe utilisateur. Pour plus d'informations, consultez Utilisation des journaux d'applications dans Visual Basic.

Voir aussi

Tâches

Comment : enregistrer des exceptions dans Visual Basic
Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log
Procédure pas à pas : modification de l'emplacement des informations My.Application.Log
Procédure pas à pas : filtrage de la sortie de My.Application.Log

Référence

My.Application.Log, objet
My.Log, objet
WriteEntry, méthode (My.Application.Log et My.Log)
WriteException, méthode (My.Application.Log et My.Log)

Concepts

Utilisation des journaux d'applications dans Visual Basic

Comment : enregistrer des exceptions dans Visual Basic

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour enregistrer des informations sur les exceptions qui se produisent dans votre application. Ces exemples indiquent comment utiliser la méthode **My.Application.Log.WriteException** pour enregistrer des exceptions que vous interceptez explicitement et des exceptions qui ne sont pas gérées.

Pour enregistrer des informations de traçage, utilisez la méthode **My.Application.Log.WriteEntry**. Pour plus d'informations, consultez WriteEntry, méthode (My.Application.Log et My.Log).

Pour enregistrer une exception gérée

1. Créez la méthode qui générera les informations sur les exceptions.

   VB

   ```
   Public Sub ExceptionLogTest(ByVal fileName As String)
   End Sub
   ```

2. Utilisez un bloc **Try...Catch** pour intercepter l'exception.

   VB

   ```
   Try
   Catch ex As Exception
   End Try
   ```

3. Placez le code susceptible de générer une exception dans le bloc **Try**.

   Supprimez les marques de commentaire des lignes `Dim` et `MsgBox` pour provoquer une exception NullReferenceException.

   VB

   ```
   ' Code that might generate an exception goes here.
   ' For example:
   '    Dim x As Object
   '    MsgBox(x.ToString)
   ```

4. Dans le bloc **Catch**, utilisez la méthode **My.Application.Log.WriteException** pour écrire les informations sur les exceptions.

   VB

   ```
   My.Application.Log.WriteException(ex, _
       TraceEventType.Error, _
       "Exception in ExceptionLogTest " & _
       "with argument " & fileName & ".")
   ```

Pour enregistrer une exception non gérée

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, choisissez **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Le fichier ApplicationEvents.vb s'ouvre.

4. Ouvrez le fichier ApplicationEvents.vb dans l'éditeur de code. Dans le menu **Général**, choisissez **Événements MyApplication**.

5. Dans le menu **Déclarations**, choisissez **UnhandledException**.

   L'application déclenche l'événement UnhandledException avant l'exécution de l'application principale.

6. Ajoutez la méthode **My.Application.Log.WriteException** au gestionnaire d'événements **UnhandledException**.

   VB

```
My.Application.Log.WriteException(e.Exception, _
    TraceEventType.Critical, _
    "Application shut down at " & _
    My.Computer.Clock.GmtTime.ToString)
```

Exemple

Cet exemple affiche le code complet pour enregistrer une exception gérée.

VB

```
Public Sub ExceptionLogTest(ByVal fileName As String)
    Try
        ' Code that might generate an exception goes here.
        ' For example:
        '    Dim x As Object
        '    MsgBox(x.ToString)
    Catch ex As Exception
        My.Application.Log.WriteException(ex, _
            TraceEventType.Error, _
            "Exception in ExceptionLogTest " & _
            "with argument " & fileName & ".")
    End Try
End Sub
```

L'exemple suivant affiche le code complet pour enregistrer une exception non gérée. Vous pouvez utiliser le **Concepteur de projets** pour accéder aux événements de l'application dans l'éditeur de code. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

VB

```
Private Sub MyApplication_UnhandledException( _
    ByVal sender As Object, _
    ByVal e As ApplicationServices.UnhandledExceptionEventArgs _
) Handles Me.UnhandledException
    My.Application.Log.WriteException(e.Exception, _
        TraceEventType.Critical, _
        "Application shut down at " & _
        My.Computer.Clock.GmtTime.ToString)
End Sub
```

Voir aussi

Tâches

Comment : écrire des messages de journal
Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log
Procédure pas à pas : modification de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet
My.Log, objet
WriteEntry, méthode (My.Application.Log et My.Log)
WriteException, méthode (My.Application.Log et My.Log)

Concepts

Utilisation des journaux d'applications dans Visual Basic

Comment : enregistrer des messages lorsque l'application démarre ou s'arrête

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour enregistrer des informations sur les événements qui se produisent dans votre application. Cet exemple indique comment utiliser la méthode **My.Application.Log.WriteEntry** avec les événements **Startup** et **Shutdown** pour écrire des informations de traçage.

Pour accéder au code du gestionnaire d'événements de l'application

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, choisissez **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Le fichier ApplicationEvents.vb s'ouvre.

Pour enregistrer des messages au démarrage de l'application

1. Ouvrez le fichier ApplicationEvents.vb dans l'éditeur de code. Dans le menu **Général**, choisissez **Événements MyApplication**.
2. Dans le menu **Déclarations**, choisissez **Démarrage**.

   L'application déclenche l'événement Startup avant l'exécution de l'application principale.
3. Ajoutez la méthode **My.Application.Log.WriteEntry** au gestionnaire d'événements **Startup**.

   VB

```
My.Application.Log.WriteEntry("Application started at " & _
    My.Computer.Clock.GmtTime.ToString)
```

Pour enregistrer des messages lorsque l'application s'arrête

1. Ouvrez le fichier ApplicationEvents.vb dans l'éditeur de code. Dans le menu **Général**, choisissez **Événements MyApplication**.
2. Dans le menu **Déclarations**, choisissez **Arrêt**.

   L'application déclenche l'événement Shutdown après l'exécution de l'application principale, mais avant qu'elle ne s'arrête.
3. Ajoutez la méthode **My.Application.Log.WriteEntry** au gestionnaire d'événements **Shutdown**.

   VB

```
My.Application.Log.WriteEntry("Application shut down at " & _
    My.Computer.Clock.GmtTime.ToString)
```

Exemple

Vous pouvez utiliser le **Concepteur de projets** pour accéder aux événements de l'application dans l'éditeur de code. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    My.Application.Log.WriteEntry("Application started at " & _
```

```
        My.Computer.Clock.GmtTime.ToString)
End Sub

Private Sub MyApplication_Shutdown( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Me.Shutdown
    My.Application.Log.WriteEntry("Application shut down at " & _
        My.Computer.Clock.GmtTime.ToString)
End Sub
```

Voir aussi

Tâches

Comment : gérer les événements d'application (Visual Basic)

Référence

My.Log, objet

My.Application.Log, objet

WriteEntry, méthode (My.Application.Log et My.Log)

WriteException, méthode (My.Application.Log et My.Log)

Concepts

Utilisation des journaux d'applications dans Visual Basic

Comment : écrire dans le journal des événements de l'application

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour écrire des informations sur les événements qui se produisent dans votre application. Cet exemple indique comment configurer un écouteur de journal des événements pour que **My.Application.Log** écrive des informations de traçage dans le journal des événements de l'application.

Vous ne pouvez pas écrire dans le journal de sécurité. Pour écrire dans le journal système, vous devez être un membre du compte LocalSystem ou Administrateur.

Pour consulter un journal des événements, vous pouvez utiliser l'**Explorateur de serveurs** ou l'**Observateur d'événements Windows**. Pour plus d'informations, consultez Comment : utiliser des journaux des événements dans l'Explorateur de serveurs ou Comment : lancer l'Observateur d'événements.

Remarque

Les journaux des événements ne sont pas pris en charge sous Windows 95, Windows 98 ou Windows Millennium.

Pour ajouter et configurer l'écouteur du journal des événements

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   ou

   S'il n'y a aucun fichier app.config,

   a. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.

   b. Dans la boîte de dialogue **Ajouter un nouvel élément**, choisissez **Fichier de configuration de l'application**.

   c. Cliquez sur **Ajouter**.

2. Recherchez la section `<listeners>` dans le fichier de configuration de l'application.

   Vous trouverez la section `<listeners>` dans la section `<source>` portant l'attribut de nom "DefaultSource" qui est imbriquée sous la section `<system.diagnostics>` elle-même imbriquée sous la section `<configuration>` de niveau supérieur.

3. Ajoutez cet élément à la section `<listeners>` :

```
<add name="EventLog"/>
```

4. Recherchez la section `<sharedListeners>` située dans la section `<system.diagnostics>` qui se trouve dans la section `<configuration>` de niveau supérieur.

5. Ajoutez cet élément à la section `<sharedListeners>` :

```
<add name="EventLog"
    type="System.Diagnostics.EventLogTraceListener, System, Version=2.0.0.0, Culture=n
eutral, PublicKeyToken=b77a5c561934e089"
     initializeData="APPLICATION_NAME"/>
```

   Remplacez `APPLICATION_NAME` par le nom de votre application.

   Remarque

   En général, une application n'écrit que des erreurs dans le journal des événements. Pour plus d'informations sur le filtrage de la sortie de journal, consultez Procédure pas à pas : filtrage de la sortie de My.Application.Log.

Pour écrire des informations sur l'événement dans le journal des événements

- Utilisez la méthode **My.Application.Log.WriteEntry** ou **My.Application.Log.WriteException** pour écrire des informations dans le journal des événements. Pour plus d'informations, consultez Comment : écrire des messages de journal et Comment : enregistrer des exceptions dans Visual Basic.

  Une fois que l'écouteur du journal des événements d'un assembly configuré, il reçoit tous les messages que **My.Applcation.Log** écrit à partir de cet assembly.

Voir aussi

Tâches

Comment : enregistrer des exceptions dans Visual Basic

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet

My.Log, objet

WriteEntry, méthode (My.Application.Log et My.Log)

WriteException, méthode (My.Application.Log et My.Log)

Concepts

Utilisation des journaux d'applications dans Visual Basic

Comment : écrire des informations sur des événements dans un fichier texte

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour enregistrer des informations sur les événements qui se produisent dans votre application. Cet exemple indique comment utiliser la méthode **My.Application.Log.WriteEntry** pour enregistrer les informations de traçage dans un fichier journal.

Pour ajouter et configurer l'écouteur de journalisation du fichier

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   ou

   S'il n'y a aucun fichier app.config :

   a. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.

   b. Dans la boîte de dialogue **Ajouter un nouvel élément**, choisissez **Fichier de configuration de l'application**.

   c. Cliquez sur **Ajouter**.

2. Recherchez la section `<listeners>` dans le fichier de configuration de l'application.

   Vous trouverez la section <écouteurs> dans la section <source> portant l'attribut de nom "DefaultSource", qui est imbriquée dans la section <system.diagnostics> elle-même imbriquée dans la section <configuration> de niveau supérieur.

3. Ajoutez cet élément à la section `<listeners>` :

```
<add name="FileLogListener" />
```

4. Recherchez la section `<sharedListeners>` situé dans la section `<system.diagnostics>` imbriquée dans la section `<configuration>` de niveau supérieur.

5. Ajoutez cet élément à la section `<sharedListeners>` :

```
<add name="FileLogListener"
    type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
          Microsoft.VisualBasic, Version=8.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"
    initializeData="FileLogListenerWriter"
    location="Custom"
    customlocation="c:\temp\" />
```

   Remplacez la valeur de l'attribut `customlocation` par le répertoire de journal.

> **Remarque**
>
> Pour définir la valeur d'une propriété d'écouteur, utilisez un attribut qui porte le même nom que la propriété, toutes les lettres du nom étant en minuscule. Par exemple, les attributs `location` et `customlocation` définissent les valeurs des pr opriétés Location et CustomLocation.

Pour écrire des informations sur l'événement dans le journal de fichier

- Utilisez la méthode **My.Application.Log.WriteEntry** ou **My.Application.Log.WriteException** pour écrire des informations dans le journal de fichier. Pour plus d'informations, consultez Comment : écrire des messages de journal et Comment : enregistrer des exceptions dans Visual Basic.

  Une fois que l'écouteur de journalisation du fichier d'un assembly configuré, il reçoit tous les messages que **My.Application.Log** écrit à partir de cet assembly.

Voir aussi

Tâches

Comment : enregistrer des exceptions dans Visual Basic

Référence

My.Application.Log, objet

My.Log, objet

WriteEntry, méthode (My.Application.Log et My.Log)

WriteException, méthode (My.Application.Log et My.Log)

Concepts

Utilisation des journaux d'applications dans Visual Basic

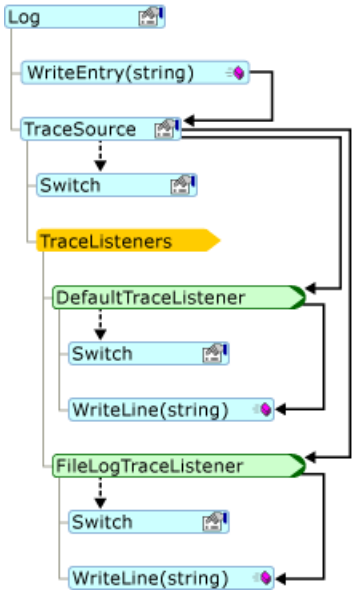
Utilisation des journaux d'applications dans Visual Basic

Les objets **My.Applicaton.Log** et **My.Log** facilitent l'écriture des informations de journalisation et de traçage dans les journaux.

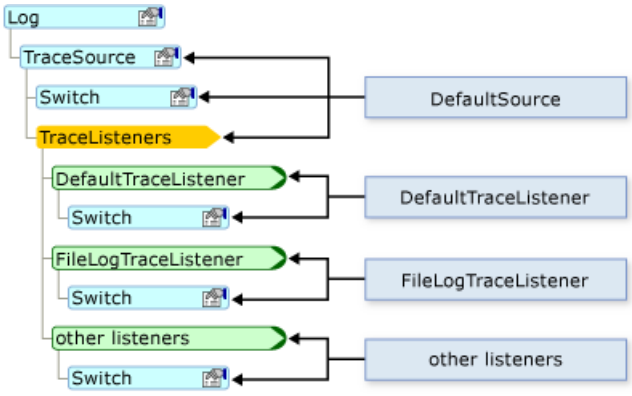
Mode d'enregistrement des messages

En premier lieu, la gravité du message est vérifiée à l'aide la propriété Switch de la propriété TraceSource du journal. Par défaut, seuls les messages contenant des « informations » de niveau grave et supérieur sont passés aux écouteurs de la trace, spécifiés dans la collection **TraceListener** du journal. Ensuite, chaque écouteur compare la gravité du message à la propriété **Switch** de l'écouteur. Si le niveau de gravité du message est assez élevé, l'écouteur écrit le message.

Le diagramme suivant montre comment un message écrit dans la méthode **WriteEntry** est passé aux méthodes **WriteLine** des écouteurs de la trace du journal :



Vous pouvez modifier le comportement du journal et des écouteurs de la trace en modifiant le fichier de configuration de l'application. Le diagramme suivant affiche la correspondance entre les parties du journal et le fichier de configuration.



Emplacement d'enregistrement des messages

Si l'assembly n'a aucun fichier de configuration, les objets **My.Application.Log** et **My.Log** écrivent dans la sortie de débogage (via la classe DefaultTraceListener) de l'application. De plus, l'objet **My.Application.Log** écrit dans le fichier journal de l'assembly (via la classe FileLogTraceListener), tandis que l'objet **My.Log** écrit dans la sortie de la page Web ASP.NET (via la classe WebPageTraceListener).

La sortie de débogage peut être affichée dans la fenêtre **Sortie** de Visual Studio lors de l'exécution de votre application en mode débogage. Pour ouvrir la fenêtre **Sortie**, cliquez sur l'élément de menu **Déboguer**, pointez sur **Windows**, puis cliquez sur **Sortie**. Dans la fenêtre **Sortie**, sélectionnez **Déboguer** dans la zone **Afficher la sortie à partir de**.

Par défaut, **My.Application.Log** écrit le fichier journal dans le chemin d'accès des données d'application de l'utilisateur. Vous pouvez obtenir le chemin d'accès dans la propriété FullLogFileName du DefaultFileLogWriter, propriété (My.Application.Log et My.Log). Le format du chemin d'accès est le suivant :

*BasePath**CompanyName**ProductName**ProductVersion*

Une valeur typique de *BasePath* se présente comme suit.

C:\Documents and Settings*username*\Application Data

Les valeurs de *CompanyName*, *ProductName*, et *ProductVersion* proviennent des informations de l'assembly de l'application. Le formulaire du nom du fichier journal est *AssemblyName*.log, *AssemblyName* étant le nom de fichier de l'assembly sans l'extension. Si plusieurs fichiers journaux sont nécessaires, par exemple si le journal d'origine n'est pas disponible lorsque l'application essaie d'écrire dans le journal, le formulaire du nom du fichier journal est *AssemblyName-iteration*.log, *iteration* étant un **Integer** positif.

Vous pouvez substituer le comportement par défaut en ajoutant ou en modifiant les fichiers de configuration de l'ordinateur et de l'application. Pour plus d'informations, consultez
Procédure pas à pas : modification de l'emplacement des informations My.Application.Log.

Configuration des paramètres du journal

L'objet **Log** a une implémentation par défaut qui fonctionne sans app.config, un fichier de configuration de l'application. Pour modifier les valeurs par défaut, vous devez ajouter un fichier de configuration dans les nouveaux paramètres. Pour plus d'informations, consultez Procédure pas à pas : filtrage de la sortie de My.Application.Log.

Les sections relatives à la configuration du journal figurent dans le nœud `<system.diagnostics>` du nœud `<configuration>` principal du fichier app.config. Les informations du journal sont définies dans plusieurs nœuds :

- Les écouteurs de l'objet **Log** sont définis dans le nœud `<sources>` nommé DefaultSource.
- Le filtre de gravité de l'objet **Log** est défini dans le nœud `<switches>` nommé DefaultSwitch.
- Les écouteurs de journalisation sont définis dans le nœud `<sharedListeners>`.

Le code suivant illustre des exemples de nœuds `<sources>`, `<switches>` et `<sharedListeners>` :

```
<configuration>
  <system.diagnostics>
    <sources>
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
        type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
          Microsoft.VisualBasic, Version=8.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL"
        initializeData="FileLogWriter"
      />
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

Modification des paramètres du journal après déploiement

Lorsque vous développez une application, ses paramètres de configuration sont stockés dans le fichier app.config, comme le montre l'exemple ci-dessus. Une fois que vous avez déployé votre application, vous pouvez toujours configurer le journal en modifiant le fichier de configuration. Dans une application Windows, le nom de ce fichier est *applicationName*.exe .config et il doit résider dans le même dossier que le fichier exécutable. Pour une application Web, il s'agit du fichier Web.config associé au projet.

Lorsque votre application exécute le code qui crée pour la première fois une instance d'une classe, elle recherche dans le fichier de configuration des informations sur l'objet. Pour l'objet **Log**, cela arrive lors de l'accès à l'objet **Log** pour la première fois. Le système examine le fichier de configuration une seule fois pour un objet particulier (la première fois que votre application crée

l'objet). Par conséquent, vous pouvez être amené à redémarrer l'application pour que les modifications prennent effet.

Dans une application déployée, vous activez le code de traçage en reconfigurant les objets commutateur avant le démarrage de votre application. Généralement, cela implique l'activation et la désactivation des objets commutateur ou le changement des niveaux de traçage, puis le redémarrage de votre application.

Considérations à propos de la sécurité

Prenez en compte les aspects suivants lorsque vous écrivez des données dans le journal :

- **Évitez les fuites d'informations utilisateur.** Vérifiez que votre application n'écrit que des informations approuvées dans le journal. Par exemple, il peut être acceptable que le journal d'applications contienne des noms d'utilisateur et non des mots de passe utilisateur.
- **Sécurisez les emplacements des journaux.** Les journaux qui contiennent des informations potentiellement sensibles doivent être stockés dans un emplacement sécurisé.
- **Évitez les informations équivoques.** En général, votre application doit valider toutes les données entrées par un utilisateur avant de les utiliser. Il s'agit notamment de l'écriture des données dans le journal d'applications.
- **Évitez le déni de service.** Si votre application écrit trop d'informations dans le journal, elle risque de remplir ce dernier et de compliquer la recherche d'informations importantes.

Voir aussi

Référence

My.Application.Log, objet

My.Log, objet

Concepts

Enregistrement d'informations provenant de l'application

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

L'objet **My.Application.Log** peut écrire des informations dans plusieurs écouteurs de journalisation. Les écouteurs de journalisation sont configurés par le fichier de configuration de l'ordinateur et peuvent être substitués par le fichier de configuration d'une application. Cette rubrique décrit les paramètres par défaut et comment déterminer les paramètres de votre application.

Pour plus d'informations sur les emplacements de sortie par défaut, consultez Utilisation des journaux d'applications dans Visual Basic.

Pour déterminer les écouteurs de My.Application.Log

1. Recherchez le fichier de configuration de l'assembly. Si vous développez l'assembly, vous pouvez accéder à app.config dans Visual Studio à partir de l'**Explorateur de solutions**. Sinon, le nom du fichier de configuration est le nom de l'assembly suivi de ".config" et figure dans le même répertoire que l'assembly.

Remarque
Tous les assemblys n'ont pas de fichier de configuration.

   Le fichier de configuration est un fichier XML.

2. Recherchez la section `<listeners>`, dans la section `<source>` avec l'attribut `name` "DefaultSource", située dans la section `<sources>`. La section `<sources>` se trouve dans la section `<system.diagnostics>`, section `<configuration>` de niveau supérieur.

   Si ces sections n'existent pas, le fichier de configuration de l'ordinateur peut configurer les écouteurs de journalisation **My.Application.Log**. Les étapes suivantes décrivent comment déterminer ce que le fichier configuration de l'ordinateur définit :

   a. Recherchez le fichier machine.config de l'ordinateur. En général, il est localisé dans le répertoire *SystemRoot*\Microsoft.NET\Framework*frameworkVersion*\CONFIG, *SystemRoot* étant le répertoire du système d'exploitation et *frameworkVersion* la version du .NET Framework.

      Les paramètres dans machine.config peuvent être substitués par le fichier de configuration d'une application.

      Si les éléments facultatifs répertoriés ci-dessous n'existez pas, vous pouvez les créer.

   b. Recherchez la section `<listeners>` dans la section `<source>` avec l'attribut `name` "DefaultSource", section `<sources>`, section `<system.diagnostics>`, section `<configuration>` de niveau supérieur.

      Si ces sections n'existent pas, le **My.Application.Log** n'a que les écouteurs de journalisation par défaut.

3. Recherchez les <éléments `add>` dans la section <`listeners`>.

   Ces éléments ajoutent les écouteurs de journalisation nommés à la source **My.Application.Log**.

4. Recherchez les éléments `<add>` portant les noms des écouteurs de journalisation dans la section `<sharedListeners>`, section `<system.diagnostics>`,section `<configuration>` de niveau supérieur.

5. Pour de nombreux types d'écouteurs partagés, les données d'initialisation de l'écouteur incluent une description de l'emplacement vers lequel l'écouteur dirige les données :

   - un écouteur Microsoft.VisualBasic.Logging.FileLogTraceListener écrit dans un journal de fichier, comme décrit dans l'introduction ;
   - un écouteur System.Diagnostics.EventLogTraceListener écrit les informations dans le journal des événements de l'ordinateur spécifié par le paramètre `initializeData`. Pour consulter un journal des événements, vous pouvez utiliser l'**Explorateur de serveurs** ou l'**Observateur d'événements Windows**. Pour plus d'informations, consultez Comment : utiliser des journaux des événements dans l'Explorateur de serveurs ou Comment : lancer l'Observateur d'événements ;

- les écouteurs System.Diagnostics.DelimitedListTraceListener et System.Diagnostics.XmlWriterTraceListener écrivent dans le fichier spécifié dans le paramètre `initializeData` ;
- un écouteur System.Diagnostics.ConsoleTraceListener écrit dans la console de ligne de commande.
- Pour plus d'informations sur l'emplacement où d'autres types d'écouteurs de journalisation écrivent des informations, consultez leur documentation.

Voir aussi

Tâches

Comment : enregistrer des exceptions dans Visual Basic
Comment : écrire des messages de journal
Procédure pas à pas : modification de l'emplacement des informations My.Application.Log
Comment : utiliser des journaux des événements dans l'Explorateur de serveurs
Comment : lancer l'Observateur d'événements
Dépannage : écouteurs de journalisation

Référence

My.Application.Log, objet
My.Log, objet
DefaultTraceListener
EventLogTraceListener
DelimitedListTraceListener
XmlWriterTraceListener
ConsoleTraceListener
System.Diagnostics

Concepts

Utilisation des journaux d'applications dans Visual Basic

Procédure pas à pas : modification de l'emplacement des informations My.Application.Log

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour enregistrer des informations sur les événements qui se produisent dans votre application. Cette procédure pas à pas indique comment substituer les paramètres par défaut pour que l'objet **Log** écrive dans d'autres écouteurs de journalisation.

Composants requis

L'objet **Log** peut écrire des informations dans plusieurs écouteurs de journalisation. Vous devez déterminer la configuration actuelle des écouteurs de journalisation avant de modifier les configurations. Pour plus d'informations, consultez Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log.

Vous pouvez examiner Comment : écrire des informations sur des événements dans un fichier texte ou Comment : écrire dans le journal des événements de l'application.

Pour ajouter des écouteurs

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   ou

   S'il n'y a aucun fichier app.config :

   a. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.

   b. Dans la boîte de dialogue **Ajouter un nouvel élément**, sélectionnez **Fichier de configuration de l'application**.

   c. Cliquez sur **Ajouter**.

2. Localisez la section `<listeners>`, sous la section `<source>` avec l'attribut `name` "DefaultSource", dans la section `<sources>`. La section `<sources>` est dans la section `<system.diagnostics>`, dans la section `<configuration>` de niveau supérieur.

3. Ajoutez ces éléments à la section `<listeners>`.

```
<!-- Uncomment to connect the application file log. -->
<!-- <add name="FileLog" /> -->
<!-- Uncomment to connect the event log. -->
<!-- <add name="EventLog" /> -->
<!-- Uncomment to connect the event log. -->
<!-- <add name="Delimited" /> -->
<!-- Uncomment to connect the XML log. -->
<!-- <add name="XmlWriter" /> -->
<!-- Uncomment to connect the console log. -->
<!-- <add name="Console" /> -->
```

4. Supprimez les marques de commentaire des écouteurs de journalisation spécifiant que vous souhaitez recevoir des messages **Log**.

5. Recherchez la section `<sharedListeners>`, dans la section `<system.diagnostics>`, dans la section `<configuration>` de niveau supérieur.

6. Ajoutez ces éléments à la section `<sharedListeners>`.

```
      <add name="FileLog"
           type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                 Microsoft.VisualBasic, Version=8.0.0.0,
                 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
           initializeData="FileLogWriter" />
      <add name="EventLog"
           type="System.Diagnostics.EventLogTraceListener,
```

```
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="sample application"/>
        <add name="Delimited"
            type="System.Diagnostics.DelimitedListTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleDelimitedFile.txt"
            delimiter=";;;"
            traceOutputOptions="DateTime" />
        <add name="XmlWriter"
            type="System.Diagnostics.XmlWriterTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleLogFile.xml" />
        <add name="Console"
            type="System.Diagnostics.ConsoleTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="true" />
```

7. Le contenu du fichier app.config doit être similaire à l'élément XML suivant :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
          <!-- Uncomment to connect the application file log. -->
          <!-- <add name="FileLog" /> -->
          <!-- Uncomment to connect the event log. -->
          <!-- <add name="EventLog" /> -->
          <!-- Uncomment to connect the event log. -->
          <!-- <add name="Delimited" /> -->
          <!-- Uncomment to connect the XML log. -->
          <!-- <add name="XmlWriter" /> -->
          <!-- Uncomment to connect the console log. -->
          <!-- <add name="Console" /> -->
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
          type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                Microsoft.VisualBasic, Version=8.0.0.0,
                Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          initializeData="FileLogWriter" />
      <add name="EventLog"
          type="System.Diagnostics.EventLogTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
          initializeData="sample application"/>
      <add name="Delimited"
```

```
            type="System.Diagnostics.DelimitedListTraceListener,
                  System, Version=2.0.0.0,
                  Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleDelimitedFile.txt"
            delimiter=";;;"
            traceOutputOptions="DateTime" />
       <add name="XmlWriter"
            type="System.Diagnostics.XmlWriterTraceListener,
                  System, Version=2.0.0.0,
                  Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleLogFile.xml" />
       <add name="Console"
            type="System.Diagnostics.ConsoleTraceListener,
                  System, Version=2.0.0.0,
                  Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="true" />
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

Pour reconfigurer un écouteur

1. Recherchez l'élément `<add>` de l'écouteur dans la section `<sharedListeners>`.
2. L'attribut `type` indique le nom du type d'écouteur. Ce type doit hériter de la classe TraceListener. Utilisez le type qui porte un nom fort pour garantir l'utilisation du type approprié. Pour plus d'informations, consultez la section « Pour référencer un type portant un nom fort » ci-dessous.

   Les types que vous pouvez utiliser sont entre autres les suivants :

   - un écouteur Microsoft.VisualBasic.Logging.FileLogTraceListener qui écrit dans un journal de fichier ;
   - un écouteur System.Diagnostics.EventLogTraceListener qui écrit des informations dans le journal des événements de l'ordinateur spécifié par le paramètre `initializeData` ;
   - les écouteurs System.Diagnostics.DelimitedListTraceListener et System.Diagnostics.XmlWriterTraceListener qui écrivent dans le fichier spécifié dans le paramètre `initializeData` ;
   - un écouteur System.Diagnostics.ConsoleTraceListener qui écrit dans la console de ligne de commande.

   Pour plus d'informations sur l'emplacement où d'autres types d'écouteurs de journalisation écrivent des informations, consultez leur documentation.
3. Lorsque l'application crée l'objet écouteur de journalisation, l'attribut `initializeData` devient le paramètre de constructeur. La signification de l'attribut `initializeData` dépend de l'écouteur de la trace.
4. Une fois qu'elle a créé l'écouteur de journalisation, l'application définit les propriétés de l'écouteur. Ces propriétés sont définies par les autres attributs dans l'élément `<add>`. Pour plus d'informations sur les propriétés d'un écouteur particulier, consultez la documentation de ce type d'écouteur.

Pour référencer un type portant un nom fort

1. Pour garantir l'utilisation du type correct pour votre écouteur de journalisation, vérifiez que vous utilisez le nom de type qualifié complet et le nom d'assemblage portant un nom fort. La syntaxe d'un type portant un nom fort se présente comme suit :

   <type name>, *<assembly name>*, *<version number>*, *<culture>*, *<strong name>*
2. Cet exemple de code indique comment déterminer le nom de type portant un nom fort pour un type qualifié complet - "System.Diagnostics.FileLogTraceListener" dans ce cas.

   VB

```
Public Sub DisplayStrongName()
```

```
    Dim t As Type = GetType(Logging.FileLogTraceListener)
    MsgBox(t.FullName & ", " & t.Assembly.FullName)
End Sub
```

Il s'agit de la sortie, et elle peut être utilisée pour référencer uniquement un type portant un nom fort, comme dans la procédure « Pour ajouter des écouteurs » ci-dessus.

```
Microsoft.VisualBasic.Logging.FileLogTraceListener, Microsoft.VisualBasic, Version=8.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
```

Voir aussi

Tâches

Comment : écrire des informations sur des événements dans un fichier texte

Comment : écrire dans le journal des événements de l'application

Référence

My.Application.Log, objet

My.Log, objet

TraceListener

Microsoft.VisualBasic.Logging.FileLogTraceListener

System.Diagnostics.EventLogTraceListener

Procédure pas à pas : filtrage de la sortie de My.Application.Log

Cette procédure pas à pas montre comment modifier le filtrage de journal par défaut de l'objet **My.Application.Log**, pour contrôler les informations passées de l'objet **Log** aux écouteurs et celles qui sont écrites par les écouteurs. Vous pouvez modifier le comportement de journalisation même après avoir généré l'application, parce que les informations de configuration sont stockées dans le fichier de configuration de l'application.

Mise en route

Chaque message écrit par le **My.Application.Log** a un niveau de gravité associé que les mécanismes de filtrage utilisent pour contrôler la sortie de journal. Cet exemple d'application utilise les méthodes **My.Application.Log** pour écrire plusieurs messages de journal avec différents niveaux de gravité.

Pour générer l'exemple d'application

1. Ouvrez un nouveau projet d'application Windows Visual Basic.
2. Ajoutez un bouton nommé Button1 à Form1.
3. Dans le gestionnaire d'événements Click de Button1, ajoutez le code suivant :

   VB

   ```
   ' Activity tracing information
   My.Application.Log.WriteEntry("Entering Button1_Click", TraceEventType.Start)

   ' Tracing information
   My.Application.Log.WriteEntry("In Button1_Click", TraceEventType.Information)

   ' Create an exception to log.
   Dim ex As New ApplicationException
   ' Exception information
   My.Application.Log.WriteException(ex)

   ' Activity tracing information
   My.Application.Log.WriteEntry("Leaving Button1_Click", TraceEventType.Stop)
   ```

4. Exécutez l'application dans le débogueur.
5. Appuyez sur **Button1**.

   L'application écrit les informations suivantes dans la sortie de débogage et le fichier journal d'applications.

   ```
   DefaultSource Information: 0 : In Button1_Click

   DefaultSource Error: 2 : Error in the application.
   ```

6. Fermez l'application.

Pour plus d'informations sur la consultation de la fenêtre de sortie de débogage de l'application, consultez Sortie, fenêtre. Pour plus d'informations sur l'emplacement du fichier journal d'applications, consultez Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log.

Remarque
Par défaut, l'application vide la sortie du fichier journal lorsque l'application se ferme.

Dans l'exemple ci-dessus, le deuxième appel au WriteEntry, méthode (My.Application.Log et My.Log) et l'appel au WriteException, méthode (My.Application.Log et My.Log) génère la sortie de journal, ce qui n'est pas le cas des premier et dernier appels à la méthode **WriteEntry**. Étant donné que les niveaux de gravité de **WriteEntry** et **WriteException** sont

« Informations »" et « Erreur », ceux-ci sont autorisés par le filtrage de journal par défaut de l'objet **My.Application.Log**. Toutefois, les événements portant le niveau de gravité « Démarrage » et « Arrêt » ne peuvent pas générer une sortie de journal.

Filtrage des écouteurs pour All My.Application.Log

L'objet **My.Application.Log** utilise un SourceSwitch nommé `DefaultSwitch` pour contrôler les messages qu'il passe des méthodes **WriteEntry** et **WriteException** aux écouteurs de journalisation. Vous pouvez configurer `DefaultSwitch` dans le fichier de configuration de l'application en affectant sa valeur à l'une des valeurs d'énumération SourceLevels. Par défaut, sa valeur est « Information ».

Ce tableau affiche le niveau de gravité requis pour le journal pour écrire un message dans les écouteurs, avec un paramètre `DefaultSwitch` particulier.

Valeur DefaultSwitch	Gravité du message requis pour la sortie
Critical	Critical
Error	**Critical** ou Error
Warning	**Critical**, **Error** ou Warning
Information	**Critical**, **Error**, **Warning** ou Information
Verbose	**Critical**, **Error**, **Warning**, **Information** ou Verbose
ActivityTracing	Start, Stop, Suspend, Resume ou Transfer
All	Tous les messages sont autorisés.
Off	Tous les messages sont bloqués.

Remarque

Les méthodes **WriteEntry** et **WriteException** ont chacune une surcharge qui ne spécifie pas de niveau de gravité. Le niveau de gravité implicite de la surcharge **WriteEntry** est « Information » et le niveau de gravité implicite de la surcharge **WriteException** est « Erreur ».

Ce tableau explique la sortie de journal affichée dans l'exemple précédent : avec le paramètre `DefaultSwitch` par défaut « Information », seul le deuxième appel à la méthode **WriteEntry** et l'appel à la méthode **WriteException** génèrent la sortie de journal.

Pour enregistrer uniquement les événements de traçage d'activités

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   - ou -

   S'il n'y a aucun fichier app.config :

   a. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.

   b. Dans la boîte de dialogue **Ajouter un nouvel élément**, choisissez **Fichier de configuration de l'application**.

   c. Cliquez sur **Ajouter**.

2. Recherchez la section `<switches>` dans la section `<system.diagnostics>` qui se trouve dans la section `<configuration>` de niveau supérieur.

3. Recherchez l'élément qui ajoute `DefaultSwitch` à la collection de commutateurs. Il doit ressembler à l'élément ci-après :

   ```
   <add name="DefaultSwitch" value="Information" />
   ```

4. Remplacez la valeur de l'attribut `value` par « ActivityTracing ».

5. Le contenu du fichier app.config doit être similaire à l'élément XML suivant :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="ActivityTracing" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
           type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                 Microsoft.VisualBasic, Version=8.0.0.0,
                 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a,
                 processorArchitecture=MSIL"
           initializeData="FileLogWriter"/>
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

6. Exécutez l'application dans le débogueur.
7. Appuyez sur **Button1**.

   L'application écrit les informations suivantes dans la sortie de débogage et le fichier journal d'applications :

   ```
   DefaultSource Start: 4 : Entering Button1_Click

   DefaultSource Stop: 5 : Leaving Button1_Click
   ```

8. Fermez l'application.
9. Remplacez la valeur de l'attribut `value` par « Information ».

Remarque

Le paramètre de configuration `DefaultSwitch` contrôle uniquement **My.Application.Log**. Il ne modifie pas le comportement des classes System.Diagnostics.Trace et System.Diagnostics.Debug du .NET Framework.

Filtrage individuel des écouteurs My.Application.Log

L'exemple précédent indique comment modifier le filtrage pour toute la sortie **My.Application.Log**. Cet exemple montre comment filtrer un écouteur de journalisation. Par défaut, une application a deux écouteurs qui écrivent dans la sortie de débogage et le fichier journal d'applications.

Le fichier de configuration contrôle le comportement des écouteurs de journalisation en permettant à chacun d'entre eux d'avoir un filtre similaire à un commutateur pour **My.Application.Log**. Un écouteur de journalisation ne génère un message que si la gravité de ce dernier est acceptée par le `DefaultSwitch` du journal et le filtre de l'écouteur de journalisation.

Cet exemple montre comment configurer le filtrage d'un nouvel écouteur de débogage et l'ajouter à l'objet **Log**. L'écouteur de débogage par défaut doit être supprimé de l'objet **Log** ; ainsi, il apparaît clairement que les messages de débogage proviennent du nouvel écouteur de débogage.

Pour enregistrer uniquement les événements de traçage d'activités

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   - ou -

S'il n'y a aucun fichier app.config :

a. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.

b. Dans la boîte de dialogue **Ajouter un nouvel élément**, choisissez **Fichier de configuration de l'application**.

c. Cliquez sur **Ajouter**.

2. Cliquez avec le bouton droit sur app.config dans **Explorateur de solutions**. Cliquez sur **Ouvrir**.

3. Recherchez la section `<listeners>` dans la section `<source>` avec l'attribut `name` « DefaultSource », situé dans la section `<sources>`. La section `<sources>` figure dans la section `<system.diagnostics>` qui se trouve dans la section `<configuration>` de niveau supérieur.

4. Ajoutez cet élément à la section `<listeners>` :

```
<!-- Remove the default debug listener. -->
<remove name="Default"/>
<!-- Add a filterable debug listener. -->
<add name="NewDefault"/>
```

5. Recherchez la section `<sharedListeners>` situé dans la section `<system.diagnostics>` qui se trouve dans la section `<configuration>` de niveau supérieur.

6. Ajoutez cet élément à la section `<sharedListeners>` :

```
<add name="NewDefault"
     type="System.Diagnostics.DefaultTraceListener,
           System, Version=2.0.0.0, Culture=neutral,
           PublicKeyToken=b77a5c561934e089,
           processorArchitecture=MSIL">
    <filter type="System.Diagnostics.EventTypeFilter"
            initializeData="Error" />
</add>
```

Le filtre EventTypeFilter considère l'une des valeurs d'énumération **SourceLevels** comme son attribut `initializeData`.

7. Le contenu du fichier app.config doit être similaire à l'élément XML suivant :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
          <!-- Remove the default debug listener. -->
          <remove name="Default"/>
          <!-- Add a filterable debug listener. -->
          <add name="NewDefault"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
           type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                 Microsoft.VisualBasic, Version=8.0.0.0,
```

```
                Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a,
                processorArchitecture=MSIL"
            initializeData="FileLogWriter"/>
        <add name="NewDefault"
            type="System.Diagnostics.DefaultTraceListener,
                System, Version=2.0.0.0, Culture=neutral,
                PublicKeyToken=b77a5c561934e089,
                processorArchitecture=MSIL">
          <filter type="System.Diagnostics.EventTypeFilter"
                  initializeData="Error" />
        </add>
      </sharedListeners>
    </system.diagnostics>
  </configuration>
```

8. Exécutez l'application dans le débogueur.
9. Appuyez sur **Button1**.

   L'application écrit les informations suivantes dans le fichier journal d'applications :

   ```
   Default Information: 0 : In Button1_Click

   Default Error: 2 : Error in the application.
   ```

   L'application écrit moins d'information dans la sortie de débogage de l'application à cause du filtrage plus restrictif.

   ```
   Default Error 2 Error
   ```

10. Fermez l'application.

Pour plus d'informations sur la modification des paramètres de journal après le déploiement, consultez Utilisation des journaux d'applications dans Visual Basic.

Voir aussi

Tâches

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log
Procédure pas à pas : modification de l'emplacement des informations My.Application.Log
Procédure pas à pas : création d'écouteurs de journalisation personnalisés
Comment : écrire des messages de journal

Concepts

Commutateurs de traçage
Enregistrement d'informations provenant de l'application

Procédure pas à pas : désactivation de la sortie My.Application.Log

Cette procédure pas à pas montre comment désactiver le filtrage de journal par défaut pour l'objet **My.Application.Log**. Vous pouvez modifier le comportement de journalisation même après avoir généré l'application, parce que les informations de configuration sont stockées dans le fichier de configuration de l'application.

Mise en route

L'objet **My.Application.Log** passe chaque message qu'il obtient à ses écouteurs de journalisation. Cet exemple d'application utilise les méthodes **My.Application.Log.WriteEntry** pour écrire un message dans les écouteurs.

Pour générer l'exemple d'application

1. Ouvrez un nouveau projet d'application Windows Visual Basic.
2. Ajoutez un bouton nommé Button1 à Form1.
3. Dans le gestionnaire d'événements Click de Button1, ajoutez le code suivant :

   VB

   ```
   My.Application.Log.WriteEntry("Log entry")
   ```

4. Exécutez l'application dans le débogueur.
5. Appuyez sur **Button1**.

   L'application écrit les informations suivantes dans la sortie de débogage et le fichier journal d'applications.

   ```
   DefaultSource Information: 0 : Log entry
   ```

6. Fermez l'application.

Pour plus d'informations sur la consultation de la fenêtre de sortie de débogage de l'application, consultez Sortie, fenêtre. Pour plus d'informations sur l'emplacement du fichier journal d'applications, consultez Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log.

Suppression d'écouteurs de My.Application.Log

Par défaut, une application a deux écouteurs qui écrivent dans la sortie de débogage et le fichier journal d'applications. Cet exemple montre comment supprimer ces écouteurs.

Pour supprimer des écouteurs de journalisation de l'objet Log

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   - ou -

   S'il n'y a aucun fichier app.config :

   a. Dans le menu **Projet**, cliquez sur **Ajouter un nouvel élément**.

   b. Dans la zone **Ajouter un nouvel élément**, choisissez **Fichier de configuration de l'application**.

   c. Cliquez sur **Ajouter**.

2. Recherchez la section `<listeners>` dans la section `<source>` portant l'attribut `name "DefaultSource"`.

   Ces sections relatives à la configuration du journal figurent dans le nœud `<system.diagnostics>` du nœud `<configuration>` principal du fichier de configuration. Le code XML de l'attribut DefaultSource figure dans le nœud `<sources>`.

3. Supprimez l'élément `<add>` portant l'attribut `name` "FileLog". Il doit ressembler à l'élément ci-après :

`<add name="FileLog"/>`

4. Ajoutez cet élément à la section `<listeners>`.

```
<!-- Remove the default debug listener. -->
<remove name="Default"/>
```

5. Le contenu du fichier app.config doit être similaire à l'élément XML suivant :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <!-- Remove the default debug listener. -->
          <remove name="Default"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
  </system.diagnostics>
</configuration>
```

6. Exécutez l'application dans le débogueur.

7. Appuyez sur **Button1**.

   L'application n'écrit pas d'informations dans le fichier journal d'applications ou dans la sortie de débogage.

Pour plus d'informations sur la modification des paramètres de journal après le déploiement, consultez Utilisation des journaux d'applications dans Visual Basic.

Voir aussi

Tâches

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Procédure pas à pas : modification de l'emplacement des informations My.Application.Log

Procédure pas à pas : filtrage de la sortie de My.Application.Log

Procédure pas à pas : création d'écouteurs de journalisation personnalisés

Comment : écrire des messages de journal

Concepts

Enregistrement d'informations provenant de l'application

Procédure pas à pas : création d'écouteurs de journalisation personnalisés

Cette procédure pas à pas illustre comment créer un écouteur de journalisation personnalisé et le configurer pour écouter la sortie de l'objet **My.Application.Log**.

Mise en route

Les écouteurs de journalisation doivent hériter de la classe TraceListener.

Pour créer l'écouteur

- Dans votre application, créez une classe nommée `SimpleListener` qui hérite de **TraceListener**.

  VB

```
Public Class SimpleListener
    Inherits System.Diagnostics.TraceListener

    <Security.Permissions.HostProtection(Synchronization:=True)> _
    Public Overloads Overrides Sub Write(ByVal message As String)
        MsgBox("Write: " & message)
    End Sub

    <Security.Permissions.HostProtection(Synchronization:=True)> _
    Public Overloads Overrides Sub WriteLine(ByVal message As String)
        MsgBox("WriteLine: " & message)
    End Sub
End Class
```

  Les méthodes Write et WriteLine, requises par la classe de base, appellent **MsgBox** pour afficher leur entrée.

  L'attribut HostProtectionAttribute est appliqué aux méthodes **Write** et **WriteLine** afin que leurs attributs correspondent aux méthodes de classe de base. L'attribut **HostProtectionAttribute** permet à l'hôte qui exécute le code de déterminer que ce dernier expose la synchronisation de la protection de l'hôte.

Remarque
L'attribut **HostProtectionAttribute** n'est effectif que sur les applications non managées, telles que SQL Server, qui hébergent le Common Language Runtime et implémentent la protection de l'hôte.

Pour vérifier que **My.Application.Log** utilise votre écouteur de journalisation, vous devez attribuer un nom fort à l'assembly qui le contient.

La procédure suivante fournit des étapes simples pour créer un assembly d'écouteur de journalisation portant un nom fort. Pour plus d'informations, consultez Création et utilisation d'assemblys avec nom fort.

Pour attribuer un nom fort à l'assembly de l'écouteur de journalisation

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, choisissez **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Signature**.
3. Sélectionnez la zone **Signer l'assembly**.
4. Sélectionnez **<Nouveau>** dans la liste déroulante **Choisir un fichier de clé de nom fort**.

   La boîte de dialogue **Créer une clé de nom fort** s'ouvre.
5. Fournissez un nom pour le fichier de clé dans la zone **Nom du fichier de clé**.

6. Entrez un mot de passe dans les zones **Entrer le mot de passe** et **Confirmer le mot de passe**.
7. Cliquez sur **OK**.
8. Régénérez l'application.

Ajout de l'écouteur

Maintenant que l'assembly a un nom fort, vous devez déterminer le nom fort de l'écouteur afin que **My.Application.Log** utilise votre écouteur de journalisation.

Le format d'un type portant un nom fort se présente comme suit.

<nom type>, <nom assembly>, <numéro version>, <culture>, < nom fort>

Pour déterminer le nom fort de l'écouteur

- Le code suivant indique comment déterminer le nom de type fort pour `SimpleListener`.

  VB

  ```
  Public Sub DisplaySimpleListenerStrongName()
      Dim t As Type = GetType(SimpleListener)
      MsgBox(t.FullName & ", " & t.Assembly.FullName)
  End Sub
  ```

  Le nom fort du type dépend de votre projet.

Avec le nom fort, vous pouvez ajouter l'écouteur à la collection de l'écouteur de journalisation **My.Application.Log**.

Pour ajouter l'écouteur à My.Application.Log

1. Cliquez avec le bouton droit sur app.config dans l'**Explorateur de solutions** et sélectionnez l'option **Ouvrir**.

   - ou -

   S'il y a un fichier app.config :

   a. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**.

   b. Dans la boîte de dialogue **Ajouter un nouvel élément**, choisissez **Fichier de configuration de l'application**.

   c. Cliquez sur **Ajouter**.

2. Recherchez la section `<listeners>`, dans la section `<source>` avec l'attribut `name` "DefaultSource", situé dans la section `<sources>`. La section `<sources>` se trouve dans la section `<system.diagnostics>`, section `<configuration>` de niveau supérieur.
3. Ajoutez cet élément à la section `<listeners>` :

   ```
   <add name="SimpleLog" />
   ```

4. Recherchez la section `<sharedListeners>`, dans la section `<system.diagnostics>`, section `<configuration>` de niveau supérieur.
5. Ajoutez cet élément à la section `<sharedListeners>` :

   ```
   <add name="SimpleLog" type="SimpleLogStrongName" />
   ```

   Modifiez la valeur de `SimpleLogStrongName` pour qu'elle soit le nom fort de l'écouteur.

Voir aussi

Tâches

Comment : enregistrer des exceptions dans Visual Basic

Comment : écrire des messages de journal

Procédure pas à pas : modification de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet

My.Log, objet

Concepts

Utilisation des journaux d'applications dans Visual Basic

Dépannage : écouteurs de journalisation

Vous pouvez utiliser les objets **My.Application.Log** et **My.Log** pour enregistrer des informations sur les événements qui se produisent dans votre application.

Pour déterminer quels écouteurs de journalisation reçoivent ces messages, consultez Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log.

L'objet **Log** peut utiliser le filtrage de journal pour limiter la quantité d'informations qu'il enregistre. Si les filtres ne sont pas correctement configurés, les journaux peuvent contenir de fausses informations. Pour plus d'informations sur le filtrage, consultez Procédure pas à pas : filtrage de la sortie de My.Application.Log.

Toutefois, même si un journal n'est pas correctement configuré, vous pouvez avoir besoin de plus d'informations sur sa configuration actuelle. Vous pouvez obtenir ces informations par le biais de la propriété **TraceSource** avancée du journal.

Pour déterminer les écouteurs de journalisation de l'objet Log dans le code

1. Importez l'espace de noms System.Diagnostics au début du fichier de code. Pour plus d'informations, consultez Imports, instruction.

   VB

   ```
   Imports System.Diagnostics
   ```

2. Créez une fonction qui retourne une chaîne qui se compose d'informations relatives à chacun des écouteurs du journal.

   VB

   ```
   Function GetListeners(ByVal listeners As TraceListenerCollection) As String
       Dim ret As String = ""
       For Each listener As TraceListener In listeners
           ret &= listener.Name
           Dim listenerType As Type = listener.GetType
           If listenerType Is GetType(DefaultTraceListener) Then
               Dim tmp As DefaultTraceListener = _
                   DirectCast(listener, DefaultTraceListener)
               ret &= ": Writes to the debug output."
           ElseIf listenerType Is GetType(Logging.FileLogTraceListener) Then
               Dim tmp As Logging.FileLogTraceListener = _
                   DirectCast(listener, Logging.FileLogTraceListener)
               ret &= ": Log filename: " & tmp.FullLogFileName
           ElseIf listenerType Is GetType(EventLogTraceListener) Then
               Dim tmp As EventLogTraceListener = _
                   DirectCast(listener, EventLogTraceListener)
               ret &= ": Event log name: " & tmp.EventLog.Log
           ElseIf listenerType Is GetType(XmlWriterTraceListener) Then
               Dim tmp As Diagnostics.XmlWriterTraceListener = _
                   DirectCast(listener, XmlWriterTraceListener)
               ret &= ": XML log"
           ElseIf listenerType Is GetType(ConsoleTraceListener) Then
               Dim tmp As ConsoleTraceListener = _
                   DirectCast(listener, ConsoleTraceListener)
               ret &= ": Console log"
           ElseIf listenerType Is GetType(DelimitedListTraceListener) Then
               Dim tmp As DelimitedListTraceListener = _
                   DirectCast(listener, DelimitedListTraceListener)
               ret &= ": Delimited log"
           Else
               ret &= ": Unhandeled log type: " & _
   ```

```
                listenerType.ToString
            End If
            ret &= vbCrLf
        Next

        Return ret
    End Function
```

3. Passez la collection des écouteurs de la trace du journal à la fonction `GetListeners` et affichez la valeur de retour.

   VB

```
Dim ListenerCollection As TraceListenerCollection
ListenerCollection = My.Application.Log.TraceSource.Listeners
Dim ListenersText As String = GetListeners(ListenerCollection)
MsgBox(ListenersText)
```

   Pour plus d'informations, consultez TraceSource, propriété (My.Application.Log et My.Log).

Voir aussi

Tâches

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet

My.Log, objet

Concepts

Utilisation des journaux d'applications dans Visual Basic

Accès à une application en cours d'exécution dans Visual Basic

Cette section contient des rubriques qui expliquent comment utiliser le modèle d'application Visual Basic pour contrôler le comportement de l'application au moment de l'exécution et des rubriques qui expliquent comment obtenir des informations relatives à l'application au moment de l'exécution.

L'objet **My.Application** fournit des propriétés, des méthodes et des événements en rapport avec l'application en cours. Il vous permet d'obtenir des informations concernant l'application, telles que la culture actuelle, la manière dont l'application a été déployée et les variables d'environnement. Le modèle d'application Visual Basic exposé par l'objet **My.Application** permet à une application d'exécuter du code lorsqu'elle démarre et s'arrête. De même, la propriété **My.Application.Deployment** fournit l'objet de déploiement ClickOnce de l'application en cours.

Tâches

Pour	Reportez-vous à
Permettre à votre application de gérer les autres événements qui peuvent être déclenchés pendant l'exécution de votre code	My.Application.DoEvents, méthode
Déterminer les arguments de ligne de commande de l'application	Comment : accéder aux arguments de ligne de commande (Visual Basic)
Vérifier si l'application a démarré avec la chaîne "/batch" comme argument	Comment : activer un mode batch pour des applications Windows Forms
Gérer toute exception non interceptée dans votre application	My.Application.UnhandledException, événement
Utiliser les événements fournis par le modèle d'application Visual Basic pour exécuter le code lorsque l'application démarre ou s'arrête.	Comment : exécuter le code lorsque l'application démarre ou se termine
Vérifier les arguments de ligne de commande des tentatives suivantes de démarrer une application à instance unique	My.Application.StartupNextInstance, événement
Télécharger et installer la version la plus récente d'une application ClickOnce	Comment : télécharger les mises à jour d'applications ClickOnce

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application, objet
My.Application.Log, objet
My.Application.Info, objet

Concepts

Vue d'ensemble du modèle d'application Visual Basic
Enregistrement d'informations provenant de l'application

Comment : exécuter le code lorsque l'application démarre ou se termine

Vous utilisez les événements fournis par le modèle d'application Visual Basic pour exécuter le code lorsque l'application démarre ou s'arrête. Le **Concepteur de projets** fournit l'accès au code du gestionnaire d'événements de l'application.

Remarque

Le modèle d'application Visual Basic n'est disponible que pour les applications Windows Forms. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour accéder au code du gestionnaire d'événements de l'application

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Le fichier ApplicationEvents.vb s'ouvre.

Pour exécuter le code au démarrage de l'application

1. Ouvrez le fichier ApplicationEvents.vb dans l'éditeur de code. Dans le menu **Général**, choisissez **Événements MyApplication**.
2. Dans le menu **Déclarations**, choisissez **Démarrage**.

   L'application déclenche l'événement Startup avant l'exécution de l'application principale.
3. Si votre application est une application à instance unique, il est utile de gérer les démarrages ultérieurs de l'application. Pour ce faire, dans le menu **Général**, choisissez **Événements MyApplication**.
4. Dans le menu **Déclarations**, choisissez **StartupNextInstance**.

   Une application à instance unique déclenche l'événement StartupNextInstance dans la première instance de l'application lorsqu'une instance suivante de l'application démarre. Ensuite, l'instance suivante se ferme sans déclencher aucun de ses propres événements.

Pour exécuter le code lorsque l'application se termine

1. Ouvrez le fichier ApplicationEvents.vb dans l'éditeur de code. Dans le menu **Général**, choisissez **Événements MyApplication**.
2. Dans le menu **Déclarations**, choisissez **Arrêt**.

   L'application déclenche l'événement Shutdown après l'exécution de l'application principale, mais avant qu'elle ne s'arrête.
3. Dans le menu **Général**, choisissez **Événements MyApplication**.
4. Dans le menu **Déclarations**, choisissez **UnhandledException**.

   L'application déclenche l'événement UnhandledException si elle rencontre une exception non gérée. Étant donné que l'application ne déclenche pas l'événement **Shutdown** après l'événement **UnhandledException**, vous souhaitez peut-être que le gestionnaire **UnhandledException** appelle le même code d'arrêt que le gestionnaire **Shutdown**.

Voir aussi

Tâches

Comment : gérer les événements d'application (Visual Basic)

Concepts

Vue d'ensemble du modèle d'application Visual Basic

Comment : activer un mode batch pour des applications Windows Forms

Cet exemple utilise l'événement **My.Application.Startup** pour vérifier si l'application a démarré avec la chaîne `/batch` comme argument.

Pour activer un mode batch pour une application Window Forms

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**.
2. Dans l'onglet **Application**, cliquez sur **Afficher les événements de l'application** pour ouvrir l'éditeur de code.
3. Créez la méthode qui gère l'événement My.Application.Startup. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

   VB
   ```
   Private Sub MyApplication_Startup( _
       ByVal sender As Object, _
       ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
   ) Handles Me.Startup
   ```

   VB
   ```
   End Sub
   ```

4. Parcourez les arguments de ligne de commande de l'application, et affectez à la propriété **Cancel** de l'objet `e` la valeur **True** si l'un des arguments est `/batch`.

   Lorsque la propriété **Cancel** a la valeur **True**, le formulaire de démarrage n'est pas lancé.

   VB
   ```
   For Each s As String In My.Application.CommandLineArgs
       If s.ToLower = "/batch" Then
           ' Stop the start form from loading.
           e.Cancel = True
       End If
   Next
   ```

5. Si la propriété **Cancel** de l'objet `e` a la valeur **True**, appelez la routine principale pour une opération sans fenêtre.

   VB
   ```
   If e.Cancel Then
       ' Call the main routine for windowless operation.
       Dim c As New BatchApplication
       c.Main()
   End If
   ```

Exemple

VB
```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
```

```
) Handles Me.Startup
    For Each s As String In My.Application.CommandLineArgs
        If s.ToLower = "/batch" Then
            ' Stop the start form from loading.
            e.Cancel = True
        End If
    Next
    If e.Cancel Then
        ' Call the main routine for windowless operation.
        Dim c As New BatchApplication
        c.Main()
    End If
End Sub
Class BatchApplication
    Sub Main()
        ' Insert code to run without a graphical user interface.
    End Sub
End Class
```

Voir aussi

Tâches

Comment : accéder aux arguments de ligne de commande (Visual Basic)

Référence

My.Application, objet

My.Application.CommandLineArgs, propriété

Concepts

Vue d'ensemble du modèle d'application Visual Basic

Comment : accéder aux arguments de ligne de commande (Visual Basic)

L'exemple de code suivant effectue une boucle sur les arguments de ligne de commande de l'application.

Exemple

VB

```
For Each argument As String In My.Application.CommandLineArgs
    ' Add code here to use the argument.
Next
```

Voir aussi

Tâches

Comment : activer un mode batch pour des applications Windows Forms

Référence

My.Application.CommandLineArgs, propriété

Structure d'un programme Visual Basic

Accès aux données de l'utilisateur

Cette section contient des rubriques abordant l'objet **My.User** et les tâches qu'il vous permet d'accomplir.

L'objet **My.User** fournit l'accès aux informations sur l'utilisateur connecté en retournant un objet qui implémente l'interface IPrincipal.

Tâches

Pour	Reportez-vous à
Obtenir le nom de connexion de l'utilisateur	Comment : déterminer le nom de connexion d'un utilisateur
Obtenir le nom de domaine de l'utilisateur si l'application utilise l'authentification Windows	Comment : déterminer le domaine de l'utilisateur
Déterminer le rôle de l'utilisateur	Comment : déterminer si un utilisateur est dans un groupe
Implémenter une authentification personnalisée	Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Voir aussi

Référence

My.User, objet

Autres ressources

Authentification et autorisation dans le .NET Framework avec Visual Basic

Comment : déterminer le nom de connexion d'un utilisateur

Vous pouvez utiliser l'objet **My.User** pour obtenir des informations sur l'utilisateur actuel. Cet exemple indique comment utiliser la propriété **My.User.Name** pour obtenir le nom de connexion de l'utilisateur.

Une application utilisant l'authentification Windows par défaut, **My.User** retourne les informations Windows sur l'utilisateur qui a démarré l'application.

Exemple

Cet exemple vérifie si l'application utilise l'authentification Windows ou personnalisée, puis utilise ces informations pour analyser la propriété **My.User.Name**.

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

Voir aussi

Tâches

Comment : déterminer le domaine de l'utilisateur

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User.Name, propriété

Concepts

Accès aux données de l'utilisateur

Comment : déterminer si un utilisateur est dans un groupe

Vous pouvez utiliser l'objet **My.User** pour obtenir des informations sur l'utilisateur actuel. Cet exemple indique comment utiliser la méthode **My.User.IsInRole** pour déterminer si l'utilisateur est un membre d'un groupe particulier.

Exemple

Cet exemple utilise la méthode **My.User.IsInRole** pour déterminer si l'utilisateur est un administrateur avant d'accéder à une ressource.

VB

```
If My.User.IsInRole( _
        ApplicationServices.BuiltInRole.Administrator) Then
    ' Insert code to access a resource here.
End If
```

Voir aussi

Tâches

Comment : déterminer le domaine de l'utilisateur

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User.IsInRole, méthode

Concepts

Accès aux données de l'utilisateur

Comment : déterminer le domaine de l'utilisateur

Vous pouvez utiliser l'objet **My.User** pour obtenir des informations sur l'utilisateur actuel. Cet exemple indique comment utiliser la propriété **My.User.Name** pour obtenir le nom de domaine de l'utilisateur si l'application utilise l'authentification Windows.

L'application utilisant l'authentification Windows par défaut, **My.User** retourne les informations Windows sur l'utilisateur qui a démarré l'application.

Exemple

Cet exemple vérifie si l'application utilise l'authentification Windows avant d'analyser la propriété **My.User.Name** pour déterminer le nom de domaine.

Cet exemple retourne une chaîne vide si l'application utilise une authentification personnalisée, parce que l'implémentation d'une authentification personnalisée ne fournit pas nécessairement des informations sur le domaine.

VB

```
Function GetUserDomain() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' My.User is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim domain As String = parts(0)
        Return domain
    Else
        ' My.User is using custom authentication.
        Return ""
    End If
End Function
```

Voir aussi

Tâches

Comment : déterminer le nom de connexion d'un utilisateur

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User.Name, propriété

Concepts

Accès aux données de l'utilisateur

Accès aux formulaires de l'application

L'objet **My.Forms** permet facilement d'accéder à une instance de chaque Windows Form déclaré dans le projet de l'application. Vous pouvez également utiliser les propriétés de l'objet **My.Application** pour accéder à l'écran de démarrage et au formulaire principal de l'application, et obtenir une liste des formulaires ouverts de l'application.

Tâches

Le tableau suivant répertorie des exemples qui indiquent comment accéder aux formulaires d'une application.

Pour	Reportez-vous à
Accéder à un formulaire à partir d'un autre formulaire d'une application.	Comment : utiliser la communication entre formulaires dans une application
Afficher les titres de tous les formulaires ouverts de l'application.	Comment : accéder à tous les formulaires ouverts d'une application
Mettre à jour l'écran de démarrage avec les informations d'état au démarrage de l'application.	My.Application.SplashScreen, propriété

Voir aussi

Référence

My.Forms, objet
My.Application.OpenForms, propriété
My.Application.SplashScreen, propriété

Comment : utiliser la communication entre formulaires dans une application

Cet exemple indique comment utiliser l'objet **My.Forms** pour ouvrir un formulaire à partir d'un autre.

L'objet **My.Forms** fournit l'accès à une instance de chaque formulaire d'une application Windows qui est déclaré dans le projet de l'application. Cela permet au code d'un formulaire de communiquer avec un autre formulaire.

Pour créer du code dans un formulaire qui peut communiquer avec un autre formulaire

1. Créez une nouvelle **Application Windows**.

   Le nom par défaut du formulaire principal est **Form1**.

   Pour plus d'informations, consultez Comment : créer un projet d'application Windows.

2. Ajoutez un autre formulaire nommé **Form2**.

   Pour plus d'informations, consultez Comment : ajouter des Windows Forms à un projet.

3. Ajoutez un bouton nommé `Button1` à **Form1** dans le Concepteur d'applications.

4. Double-cliquez sur **Button1**.

5. Dans le gestionnaire d'événements de l'événement Click de `Button1`, ajoutez le code suivant :

   VB

   ```
   My.Forms.Form2.Text = Now.ToString
   My.Forms.Form2.Show()
   ```

6. Exécutez l'application.

7. Cliquez sur **Button1**.

   Form2 s'ouvre et son titre affiche l'heure actuelle.

8. Patientez pendant quelques secondes et cliquez sur **Button1** une nouvelle fois.

   Form2 reste ouvert et son titre se met à jour à l'heure actuelle.

Voir aussi

Tâches

Comment : accéder à tous les formulaires ouverts d'une application

Référence

My.Forms, objet

Concepts

Accès aux formulaires de l'application

Comment : accéder à tous les formulaires ouverts d'une application

Cet exemple utilise la propriété **My.Application.OpenForms** pour afficher les titres de tous les formulaires ouverts de l'application.

La propriété **My.Application.OpenForms** retourne tous les formulaires actuellement ouverts quel que soit le thread qui les a ouverts. Cela signifie que vous devez vérifier la propriété InvokeRequired de chaque formulaire avant d'y accéder ; sinon, elle peut lever une exception InvalidOperationException.

Cet exemple déclare une fonction qui obtient le titre de chaque formulaire en mode thread-safe. En premier lieu, il vérifie la propriété **InvokeRequired** du formulaire et, si nécessaire, utilise la méthode BeginInvoke pour exécuter la fonction sur le thread du formulaire. Ensuite, la fonction retourne le titre du formulaire.

Exemple

Cet exemple effectue une boucle sur les formulaires ouverts de l'application et affiche leurs titres dans un contrôle ListBox. Pour un code plus simple qui n'affiche que les formulaires directement accessibles à partir du thread actuel, consultez My.Application.OpenForms, propriété.

VB

```
Private Sub GetOpenFormTitles()
    Dim formTitles As New Collection

    Try
        For Each f As Form In My.Application.OpenForms
            ' Use a thread-safe method to get all form titles.
            formTitles.Add(GetFormTitle(f))
        Next
    Catch ex As Exception
        formTitles.Add("Error: " & ex.Message)
    End Try

    Form1.ListBox1.DataSource = formTitles
End Sub

Private Delegate Function GetFormTitleDelegate(ByVal f As Form) As String
Private Function GetFormTitle(ByVal f As Form) As String
    ' Check if the form can be accessed from the current thread.
    If Not f.InvokeRequired Then
        ' Access the form directly.
        Return f.Text
    Else
        ' Marshal to the thread that owns the form.
        Dim del As GetFormTitleDelegate = AddressOf GetFormTitle
        Dim param As Object() = {f}
        Dim result As System.IAsyncResult = f.BeginInvoke(del, param)
        ' Give the form's thread a chance process function.
        System.Threading.Thread.Sleep(10)
        ' Check the result.
        If result.IsCompleted Then
            ' Get the function's return value.
            Return "Different thread: " & f.EndInvoke(result).ToString
        Else
            Return "Unresponsive thread"
        End If
    End If
End Function
```

Voir aussi

Tâches

Comment : faire des appels thread-safe aux contrôles Windows Forms

Référence

My.Application.OpenForms, propriété

Accès aux services Web d'une application

L'objet **My.WebServices** fournit une instance de chaque service Web référencé par le projet actuel. Chaque instance est instanciée sur demande. Vous pouvez accéder à ces services Web via les propriétés de l'objet **My.WebServices**. La propriété porte le même nom que celui du service Web auquel la propriété accède. Toute classe qui hérite de SoapHttpClientProtocol est un service Web. Pour plus d'informations sur l'ajout de services Web à un projet, consultez Comment : accéder à un service Web XML en code managé.

Tâches

Le tableau suivant répertorie les moyens possibles pour accéder aux services Web référencées par une application.

Pour	Reportez-vous à
Appeler un service Web	My.WebServices, objet
Appeler un service Web de façon asynchrone et gérer un événement lorsqu'il s'exécute	Comment : appeler un service Web de manière asynchrone

Voir aussi

Référence

My.WebServices, objet

Comment : appeler un service Web de manière asynchrone

Cet exemple joint un gestionnaire à l'événement de gestionnaire asynchrone d'un service Web XML, afin qu'il puisse récupérer le résultat d'un appel de méthode asynchrone. Cet exemple a utilisé le service Web DemoTemperatureService sur http://www.xmethods.net.

Lorsque vous référencez un service Web de votre projet dans l'environnement de développement intégré (IDE) Visual Studio, il est ajouté à l'objet **My.WebServices** et l'environnement de développement intégré génère une classe de proxy client pour accéder à un service Web spécifié.

La classe proxy vous permet d'appeler les méthodes Web Service de façon synchrone, là où votre application attend la fin de la fonction. De plus, le proxy crée des membres supplémentaires pour aider à appeler la méthode de façon asynchrone. Pour chaque fonction de service Web, *NameOfWebServiceFunction*, le proxy crée une sous-routine *NameOfWebServiceFunction***Async**, un événement *NameOfWebServiceFunction***Completed** et une classe *NameOfWebServiceFunction***CompletedEventArgs**. Cet exemple montre comment utiliser les membres asynchrones pour accéder à la fonction `getTemp` du service Web DemoTemperatureService.

Remarque

Ce code ne fonctionne pas dans les applications Web parce qu'ASP.NET ne prend pas en charge l'objet **My.WebServices**.

Pour appeler un service Web de façon asynchrone

1. Référencez le service Web DemoTemperatureService sur http://www.xmethods.net. L'adresse est

   ```
   http://www.xmethods.net/sd/2001/DemoTemperatureService.wsdl
   ```

   Pour plus d'informations, consultez Comment : accéder à un service Web XML en code managé.

2. Ajoutez un gestionnaire d'événements pour l'événement `getTempCompleted` :

   ```
   Private Sub getTempCompletedHandler(ByVal sender As Object, _
       ByVal e As net.xmethods.www.getTempCompletedEventArgs)

       MsgBox("Temperature: " & e.Result)
   End Sub
   ```

   Remarque

   Vous ne pouvez pas utiliser l'instruction **Handles** pour associer un gestionnaire d'événements aux événements de l'obj et **My.WebServices**.

3. Ajoutez un champ à suivre si le gestionnaire d'événements a été ajouté à l'événement `getTempCompleted`:

   ```
   Private handlerAttached As Boolean = False
   ```

4. Ajoutez une méthode pour ajouter le gestionnaire d'événements à l'événement `getTempCompleted`, si nécessaire, et pour appeler la méthode `getTempAsynch` :

   ```
   Sub CallGetTempAsync(ByVal zipCode As Integer)
       If Not handlerAttached Then
           AddHandler _
               My.WebServices.TemperatureService.getTempCompleted, _
               AddressOf Me.TS_getTempCompleted
           handlerAttached = True
       End If
   ```

```
    My.WebServices.TemperatureService.getTempAsync(zipCode)
End Sub
```

Pour appeler de façon asynchrone la méthode Web `getTemp`, appelez la méthode `CallGetTempAsync`. Lorsque la méthode Web prend fin, sa valeur de retour est passée au gestionnaire d'événements `getTempCompletedHandler`.

Voir aussi

Tâches

Comment : accéder à un service Web XML en code managé

Référence

My.WebServices, objet

Concepts

Accès aux services Web d'une application

Accès aux ressources de l'application

Cette section contient des pages d'aide abordant l'objet **My.Resources** et les tâches que vous pouvez accomplir à l'aide de l'objet **My.Resources**.

Les propriétés de l'objet **My.Resources** fournissent un accès en lecture seule aux ressources de votre application.

Tâches

Le tableau suivant répertorie des exemples qui indiquent comment accéder aux ressources d'une application.

Pour	Reportez-vous à
Récupérer une ressource audio	Comment : récupérer des ressources audio dans Visual Basic
Récupérer une ressource d'icône	Comment : récupérer des ressources de type icône dans Visual Basic
Récupérer une ressource d'image	Comment : récupérer des ressources de type image dans Visual Basic
Récupérer une ressource de type chaîne	Comment : récupérer des ressources de type chaîne dans Visual Basic
Récupérer une ressource localisée	Comment : récupérer des ressources localisées dans Visual Basic

Voir aussi

Référence

My.Resources, objet

Autres ressources

Gestion des ressources de l'application

Comment : récupérer des ressources audio dans Visual Basic

Vous pouvez récupérer les ressources audio d'une application en accédant à la propriété de la ressource dans l'objet **My.Resources**.

L'objet **My.Resources** expose chaque ressource en tant que propriété en lecture seule. Le nom de la propriété est le même que le nom de la ressource, et le type de propriété est déterminé par la classification de la ressource. Pour plus d'informations, consultez My.Resources, objet.

Exemple

Cet exemple utilise la méthode **My.Computer.Audio.Play** pour lire le son stocké comme une ressource audio nommée `Form1Greeting` dans le fichier de ressources de l'application.

VB

```
Sub PlayFormGreeting()
    My.Computer.Audio.Play(My.Resources.Form1Greeting, _
        AudioPlayMode.Background)
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir une ressource audio nommée `Form1Greeting` dans le fichier de ressources de l'application. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

La méthode **My.Computer.Audio.Play** n'est disponible que pour les applications Windows Forms. Pour plus d'informations, consultez My.Computer.Audio.Play, méthode.

Voir aussi

Tâches

Comment : récupérer des ressources de type chaîne dans Visual Basic
Comment : récupérer des ressources de type image dans Visual Basic
Comment : récupérer des ressources de type icône dans Visual Basic
Comment : récupérer des ressources localisées dans Visual Basic

Référence

My.Resources, objet
My.Computer.Audio.Play, méthode

Autres ressources

Gestion des ressources de l'application

Comment : récupérer des ressources de type icône dans Visual Basic

Vous pouvez récupérer les ressources d'icône d'une application en accédant à la propriété de la ressource dans l'objet **My.Resources**.

L'objet **My.Resources** expose chaque ressource en tant que propriété en lecture seule. Le nom de la propriété est le même que le nom de la ressource, et le type de propriété est déterminé par la classification de la ressource. Pour plus d'informations, consultez My.Resources, objet.

Exemple

Cet exemple affecte l'icône du formulaire à l'icône nommée `Form1Icon` dans le fichier de ressources de l'application stockée.

VB

```
Sub SetFormIcon()
    Me.Icon = My.Resources.Form1Icon
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir une icône nommée `Form1Icon` dans le fichier de ressources de l'application. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Voir aussi

Tâches

Comment : récupérer des ressources de type chaîne dans Visual Basic
Comment : récupérer des ressources de type image dans Visual Basic
Comment : récupérer des ressources audio dans Visual Basic
Comment : récupérer des ressources localisées dans Visual Basic

Référence

My.Resources, objet

Autres ressources

Gestion des ressources de l'application

Comment : récupérer des ressources de type image dans Visual Basic

Vous pouvez récupérer les ressources d'image d'une application en accédant à la propriété de la ressource dans l'objet **My.Resources**.

L'objet **My.Resources** expose chaque ressource en tant que propriété en lecture seule. Le nom de la propriété est le même que le nom de la ressource, et le type de propriété est déterminé par la classification de la ressource. Pour plus d'informations, consultez My.Resources, objet.

Exemple

Cet exemple affecte l'image d'arrière-plan du formulaire à la ressource d'image nommée `Form1Background` qui figure dans le fichier de ressources de l'application.

VB

```
Sub SetFormBackgroundImage()
    Me.BackgroundImage = My.Resources.Form1Background
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir une ressource d'image nommée `Form1Background` dans le fichier de ressources de l'application. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Voir aussi

Tâches

Comment : récupérer des ressources de type chaîne dans Visual Basic
Comment : récupérer des ressources de type icône dans Visual Basic
Comment : récupérer des ressources audio dans Visual Basic
Comment : récupérer des ressources localisées dans Visual Basic

Référence

My.Resources, objet

Autres ressources

Gestion des ressources de l'application

Comment : récupérer des ressources de type chaîne dans Visual Basic

Vous pouvez récupérer les ressources de type chaîne d'une application en accédant à la propriété de la ressource dans l'objet **My.Resources**.

L'objet **My.Resources** expose chaque ressource en tant que propriété en lecture seule. Le nom de la propriété est le même que le nom de la ressource, et le type de propriété est déterminé par la classification de la ressource. Pour plus d'informations, consultez My.Resources, objet.

Exemple

Cet exemple affecte au titre du formulaire la ressource de type chaîne nommée `Form1Title` à partir du fichier de ressources de l'application.

VB

```
Sub SetFormTitle()
    Me.Text = My.Resources.Form1Title
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir une chaîne nommée `Form1Title` dans le fichier de ressources de l'application. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Voir aussi

Tâches

Comment : récupérer des ressources de type image dans Visual Basic
Comment : récupérer des ressources de type icône dans Visual Basic
Comment : récupérer des ressources audio dans Visual Basic
Comment : récupérer des ressources localisées dans Visual Basic

Référence

My.Resources, objet

Autres ressources

Gestion des ressources de l'application

Comment : récupérer des ressources localisées dans Visual Basic

L'objet **My.Resources** fournit les ressources d'application localisées (si disponible) en fonction des paramètres de culture de l'ordinateur sur lequel s'exécute l'application. Vous pouvez substituer les paramètres de culture de l'ordinateur en définissant la propriété My.Application.UICulture.

Le runtime identifie une ressource localisée par la signature de sa culture, ou nom. Resources.resx, le fichier de ressources par défaut du projet, détermine les propriétés que l'objet **My.Resources** affiche. Pour fournir des ressources localisées, vous devez :

- copier et attribuer au fichier de ressources le nouveau nom Resources.*CultureSignature*.resx ;
- rechercher les chaînes et les fichiers référencés par le fichier de ressources ;
- ajouter le fichier de ressources à votre projet.

L'objet **My.Resources** expose chaque ressource en tant que propriété en lecture seule. Le nom de la propriété est le même que le nom de la ressource, et le type de propriété est déterminé par la classification de la ressource. Pour plus d'informations, consultez My.Resources, objet et Ressources dans les applications.

Chaque culture a un nom unique, qui est une combinaison d'un nom de culture de deux lettres en minuscules associé à une langue, et si nécessaire, un nom de sous-culture en majuscules à deux lettres associé à un pays ou une région. Le nom de la sous-culture suit le nom de la culture, séparé par un tiret (-). Par exemple, ja-JP pour japonais au Japon, en-US pour anglais américain ou de-DE pour allemand en Allemagne (par opposition à de-AT pour allemand en Autriche). Pour plus d'informations sur les noms de cultures, consultez CultureInfo.

Exemple

Cet exemple récupère la version de culture française de la ressource de type chaîne de l'application nommée `Message`.

Pour modifier la culture utilisée par l'objet **My.Resources**, cet exemple utilise la méthode My.Application.ChangeUICulture.

VB

```
Sub ShowLocalizedMessage()
    Dim culture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.Message)
    My.Application.ChangeUICulture(culture)
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir une chaîne nommée `Message` dans le fichier de ressources de l'application et cette dernière doit avoir la version de la culture française de ce fichier de ressources, Resources.fr-FR.resx. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Si l'application n'a pas la version de la culture française de ce fichier de ressources, l'objet **My.Resource** récupère la ressource à partir du fichier de ressources de la culture par défaut.

Voir aussi

Tâches

Comment : récupérer des ressources de type chaîne dans Visual Basic

Référence

My.Resources, objet

Autres ressources

Gestion des ressources de l'application

Accès aux paramètres d'application

Cette section contient des rubriques qui décrivent l'objet **My.Settings** et les tâches qu'il vous permet d'accomplir.

My.Settings

Les propriétés de l'objet **My.Settings** fournissent l'accès aux paramètres de votre application. Pour ajouter ou supprimer des paramètres, utilisez le **Concepteur de projets**. Pour plus d'informations, consultez Comment : ajouter ou supprimer des paramètres d'application.

Les méthodes de l'objet **My.Settings** vous permettent d'enregistrer les paramètres utilisateur actuels ou restaurer les dernières valeurs enregistrées.

Tâches

Le tableau suivant répertorie des exemples qui indiquent comment accéder aux formulaires d'une application.

Pour	Reportez-vous à
Mettre à jour la valeur d'un paramètre utilisateur	Comment : modifier les paramètres utilisateur dans Visual Basic
Afficher les paramètres d'application et utilisateur dans une grille de propriétés	Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic
Enregistrer les valeurs des paramètres utilisateur mis à jour	Comment : rendre persistants les paramètres utilisateur dans Visual Basic
Déterminer les valeurs des paramètres utilisateur	Comment : lire des paramètres d'application dans Visual Basic

Voir aussi

Référence

My.Settings, objet

Autres ressources

Gestion des paramètres de l'application

Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic

Vous pouvez créer une grille de propriétés pour les paramètres utilisateur en remplissant un contrôle PropertyGrid à l'aide des propriétés des paramètres utilisateur de l'objet **My.Settings**.

Remarque

Pour que cet exemple fonctionne, vous devez configurer les paramètres utilisateur de votre application. Pour plus d'informations sur l'ajout de paramètres utilisateur, consultez Comment : ajouter ou supprimer des paramètres d'application.

L'objet **My.Settings** expose chaque paramètre comme une propriété. Le nom de la propriété et celui du paramètre sont identiques, de même que le type de propriété et le type de paramètre. La **Portée** du paramètre détermine si la propriété est en lecture seule : la propriété d'un paramètre de portée **Application** est en lecture seule, tandis que la propriété d'un paramètre de portée **Utilisateur** est en mode lecture-écriture. Pour plus d'informations, consultez My.Settings, objet.

Remarque

Vous ne pouvez pas modifier ni enregistrer les valeurs des paramètres de portée application au moment de l'exécution. Les paramètres de portée application ne peuvent être modifiés que lors de la création de l'application (via le **Concepteur de projets**) ou de la modification du fichier de configuration de l'application. Pour plus d'informations, consultez Gestion des paramètres de l'application.

Cet exemple utilise un contrôle **PropertyGrid** pour accéder aux propriétés des paramètres utilisateur de l'objet **My.Settings**. Par défaut, la **PropertyGrid** affiche toutes les propriétés de l'objet **My.Settings**. Toutefois, les propriétés des paramètres utilisateur ont l'attribut UserScopedSettingAttribute. Cet exemple affecte à la propriété BrowsableAttributes du **PropertyGrid** la valeur **UserScopedSettingAttribute** pour afficher uniquement les propriétés des paramètres utilisateur.

Pour ajouter une grille de propriétés de paramètres utilisateur

1. Ajoutez le contrôle **PropertyGrid** de la **Boîte à outils** à l'aire de conception de votre application, représentée dans cet exemple par `Form1`.

   Le nom par défaut du contrôle de la grille de propriétés est `PropertyGrid1`.

2. Double-cliquez sur l'aire de conception de `Form1` pour ouvrir le code pour le gestionnaire d'événements de chargement du formulaire.

3. Définissez l'objet **My.Settings** comme étant l'objet sélectionné pour la grille de propriétés.

   VB

   ```
   PropertyGrid1.SelectedObject = My.Settings
   ```

4. Configurez la grille de propriétés pour afficher uniquement les paramètres utilisateur.

   VB

   ```
   ' Attribute for the user-scope settings.
   Dim userAttr As New System.Configuration.UserScopedSettingAttribute
   Dim attrs As New System.ComponentModel.AttributeCollection(userAttr)
   PropertyGrid1.BrowsableAttributes = attrs
   ```

   Remarque

   Pour afficher uniquement les paramètres de portée application, utilisez l'attribut ApplicationScopedSettingAttribute au lieu de **UserScopedSettingAttribute**.

Programmation fiable

L'application enregistre les paramètres utilisateur lorsqu'elle s'arrête. Pour enregistrer les paramètres immédiatement, appelez la méthode **My.Settings.Save**. Pour plus d'informations, consultez Comment : rendre persistants les paramètres utilisateur dans Visual Basic.

Voir aussi

Tâches

Comment : lire des paramètres d'application dans Visual Basic
Comment : modifier les paramètres utilisateur dans Visual Basic
Comment : rendre persistants les paramètres utilisateur dans Visual Basic
Comment : ajouter ou supprimer des paramètres d'application

Référence

My.Settings, objet

Autres ressources

Gestion des paramètres de l'application

Comment : rendre persistants les paramètres utilisateur dans Visual Basic

Vous pouvez utiliser la méthode **My.Settings.Save** pour rendre persistantes les modifications apportées aux paramètres utilisateur.

En général, les applications sont conçues pour rendre persistantes les modifications apportées aux paramètres utilisateur lorsque l'application s'arrête. La raison est que l'enregistrement des paramètres peut prendre quelques secondes, en fonction de nombreux facteurs.

Pour plus d'informations, consultez My.Settings, objet.

Remarque

Même si vous pouvez modifier et enregistrer les valeurs des paramètres de portée utilisateur au moment de l'exécution, les paramètres de portée application sont en lecture seule et ne peuvent pas être modifiés par programme. Vous pouvez modifier des paramètres de portée application lors de la création de l'application, à l'aide du **Concepteur de projets** ou en modifiant le fichier de configuration de l'application. Pour plus d'informations, consultez Gestion des paramètres de l'application.

Exemple

Cet exemple modifie la valeur du paramètre utilisateur `LastChanged` et enregistre cette modification en appelant la méthode **My.Settings.Save**.

VB

```
Sub ChangeAndPersistSettings()
    My.Settings.LastChanged = Today
    My.Settings.Save()
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir un paramètre utilisateur `LastChanged` de type **Date**. Pour plus d'informations, consultez Comment : ajouter ou supprimer des paramètres d'application.

Voir aussi

Tâches

Comment : lire des paramètres d'application dans Visual Basic
Comment : modifier les paramètres utilisateur dans Visual Basic
Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic
Comment : ajouter ou supprimer des paramètres d'application

Référence

My.Settings, objet

Autres ressources

Gestion des paramètres de l'application

Comment : modifier les paramètres utilisateur dans Visual Basic

Vous pouvez modifier un paramètre utilisateur en assignant une nouvelle valeur à la propriété du paramètre de l'objet **My.Settings**.

L'objet **My.Settings** expose chaque paramètre comme une propriété. Le nom de la propriété et celui du paramètre sont identiques, de même que le type de propriété et le type de paramètre. La **Portée** du paramètre détermine si la propriété est en lecture seule : la propriété d'un paramètre de portée **Application** est en lecture seule, tandis que la propriété d'un paramètre de portée **Utilisateur** est en mode lecture-écriture. Pour plus d'informations, consultez My.Settings, objet.

Remarque

Même si vous pouvez modifier et enregistrer les valeurs des paramètres de portée utilisateur au moment de l'exécution, les paramètres de portée application sont en lecture seule et ne peuvent pas être modifiés par programme. Vous pouvez modifier des paramètres de portée application lors de la création de l'application, à l'aide du **Concepteur de projets** ou en modifiant le fichier de configuration de l'application. Pour plus d'informations, consultez Gestion des paramètres de l'application.

Exemple

Cet exemple modifie la valeur du paramètre utilisateur `Nickname`.

VB

```
Sub ChangeNickname(ByVal newNickname As String)
    My.Settings.Nickname = newNickname
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir un paramètre utilisateur `Nickname` de type **String**. Pour plus d'informations, consultez Comment : ajouter ou supprimer des paramètres d'application.

L'application enregistre les paramètres utilisateur lorsque l'application s'arrête. Pour enregistrer les paramètres immédiatement, appelez la méthode **My.Settings.Save**. Pour plus d'informations, consultez Comment : rendre persistants les paramètres utilisateur dans Visual Basic.

Voir aussi

Tâches

Comment : lire des paramètres d'application dans Visual Basic
Comment : rendre persistants les paramètres utilisateur dans Visual Basic
Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic
Comment : ajouter ou supprimer des paramètres d'application

Référence

My.Settings, objet

Autres ressources

Gestion des paramètres de l'application

Comment : lire des paramètres d'application dans Visual Basic

Vous pouvez lire un paramètre utilisateur en accédant à la propriété du paramètre de l'objet **My.Settings**.

L'objet **My.Settings** expose chaque paramètre comme une propriété. Le nom de la propriété et celui du paramètre sont identiques, de même que le type de propriété et le type de paramètre. La **Portée** du paramètre détermine si la propriété est en lecture seule : la propriété d'un paramètre de portée **Application** est en lecture seule, tandis que la propriété d'un paramètre de portée **Utilisateur** est en mode lecture-écriture. Pour plus d'informations, consultez My.Settings, objet.

Exemple

Cet exemple affiche la valeur du paramètre `Nickname`.

VB

```
Sub ShowNickname()
    MsgBox("Nickname is " & My.Settings.Nickname)
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir un paramètre `Nickname` de type **String**. Pour plus d'informations, consultez Comment : ajouter ou supprimer des paramètres d'application.

Voir aussi

Tâches

Comment : modifier les paramètres utilisateur dans Visual Basic
Comment : rendre persistants les paramètres utilisateur dans Visual Basic
Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic
Comment : ajouter ou supprimer des paramètres d'application

Référence

My.Settings, objet

Autres ressources

Gestion des paramètres de l'application

Traitement de lecteurs, de répertoires et de fichiers

Vous pouvez utiliser Visual Basic pour traiter des lecteurs, des dossiers et des fichiers avec l'objet **My.Computer.FileSystem** qui fournit de meilleures performances et est plus facile à utiliser que les méthodes traditionnelles, telles que les fonctions **FileOpen** et **Write** (bien qu'elles soient encore disponibles). Ces méthodes sont traitées en détail dans les sections suivantes.

Dans cette section

Accès au fichier avec Visual Basic

Décrit l'utilisation de l'objet **My.Computer.FileSystem** pour travailler avec des fichiers, des lecteurs et des dossiers.

Procédure pas à pas : manipulation de fichiers à l'aide de méthodes du .NET Framework

Explique comment utiliser le .NET Framework pour manipuler des fichiers et des dossiers.

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Explique comment utiliser l'objet **My.Computer.FileSystem** pour manipuler des fichiers et des dossiers.

Rubriques connexes

Structure de programme et conventions de codage

Fournit des indications relatives à la structure physique et à l'apparence des programmes.

My.Computer.FileSystem, objet

Fournit une documentation de référence pour l'objet **My.Computer.FileSystem** et ses membres.

E/S de fichier dans le .NET Framework avec Visual Basic

Fournit une vue d'ensemble des E/S de fichier avec le .NET Framework.

Accès au fichier avec Visual Basic

L'objet **My.Computer.FileSystem** fournit des outils pour utiliser les fichiers et les dossiers. Ses propriétés, méthodes et événements vous permettent de créer, copier, déplacer, étudier et supprimer des fichiers et des dossiers. **My.Computer.FileSystem** fournit de meilleures performances que les fonctions héritées (**FileOpen**, **FileClose**, **Input**, **InputString**, **LineInput**, etc.) fournies par Visual Basic pour la compatibilité descendante.

Dans cette section

Lecture à partir de fichiers en Visual Basic

Répertorie les rubriques qui décrivent l'utilisation de l'objet **My.Computer.FileSystem** pour lire les fichiers.

Écriture dans des fichiers en Visual Basic

Répertorie les rubriques qui décrivent l'utilisation de l'objet **My.Computer.FileSystem** pour écrire dans les fichiers.

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Répertorie les rubriques qui décrivent l'utilisation de l'objet **My.Computer.FileSystem** pour créer, copier, supprimer et déplacer des fichiers et des dossiers.

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Répertorie les rubriques qui décrivent l'utilisation de l'objet **My.Computer.FileSystem** pour déterminer les propriétés des fichiers, des dossiers et des lecteurs.

Analyse des fichiers texte avec l'objet TextFieldParser

Explique comment utiliser **TextFieldReader** pour analyser des fichiers texte tels que des journaux.

Codages de fichiers

Décrit les codages de fichier et leur utilisation.

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Explique comment créer un utilitaire qui rapporte des informations concernant les fichiers et les dossiers.

Dépannage : lecture et écriture dans des fichiers texte

Répertorie les problèmes couramment rencontrés lors de la lecture et de l'écriture dans des fichiers texte et suggère des solutions pour chaque problème.

Lecture à partir de fichiers en Visual Basic

Cette section répertorie les tâches associées à la lecture de fichiers.

Dans cette section

Comment : lire des fichiers texte dans Visual Basic

Montre comment lire un fichier texte.

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Montre comment lire un fichier texte délimité.

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Montre comment lire un fichier texte à largeur fixe.

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Montre comment lire un fichier texte avec plusieurs formats.

Comment : lire des fichiers binaires dans Visual Basic

Montre comment lire un fichier texte binaire.

Comment : lire des fichiers texte existants dans Mes documents (Visual Basic)

Montre comment lire un fichier texte dans le répertoire **Mes documents**.

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

Montre comment utiliser un StreamReader pour lire un fichier.

Référence

My.Computer.FileSystem, objet

Décrit l'objet **My.Computer.FileSystem** et ses membres.

My.Computer.FileSystem.ReadAllText, méthode

Décrit la méthode **ReadAllText**.

My.Computer.FileSystem.ReadAllBytes, méthode

Décrit la méthode **ReadAllBytes**.

My.Computer.FileSystem.OpenTextFieldParser, méthode

Décrit la méthode **OpenTextFieldParser**.

My.Computer.FileSystem.OpenTextFileReader, méthode

Décrit la méthode **OpenTextFileReader**.

Rubriques connexes

Comment : lire le Presse-papiers en Visual Basic

Explique comment lire des données contenues dans le Presse-papiers.

Analyse des fichiers texte avec l'objet TextFieldParser

Fournit une vue d'ensemble de la lecture de fichiers texte avec l'objet **TextFieldParser**.

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Montre comment utiliser la fonctionnalité **My** avec les fichiers et les répertoires.

Procédure pas à pas : manipulation de fichiers à l'aide de méthodes du .NET Framework

Montre comment utiliser les méthodes du .NET Framework avec les fichiers et les répertoires.

Comment : lire des fichiers texte dans Visual Basic

La méthode **ReadAllText** de l'objet **My.Computer.FileSystem** vous permet de lire un fichier texte. Le codage du fichier peut être spécifié si le contenu de ce dernier utilise le codage ASCII ou UTF-8.

Si vous lisez un fichier avec des caractères étendus, vous devez spécifier son codage.

Pour lire un fichier texte

- Utilisez la méthode **ReadAllText** de l'objet **My.Computer.FileSystem** pour lire le contenu d'un fichier texte dans une chaîne en fournissant le chemin d'accès. L'exemple suivant lit le contenu du fichier test.txt dans une chaîne puis l'affiche dans un message.

  VB

```
Dim fileReader As String
fileReader = My.Computer.FileSystem.ReadAllText("C:\test.txt")
MsgBox(fileReader)
```

Pour lire un fichier texte codé

- Utilisez la méthode **ReadAllText** de l'objet **My.Computer.FileSytem** pour lire le contenu d'un fichier texte dans une chaîne en fournissant le chemin d'accès et le type de codage du fichier. L'exemple suivant lit le contenu du fichier UTF32 test.txt dans une chaîne puis l'affiche dans un message.

  VB

```
Dim fileReader As String
fileReader = My.Computer.FileSystem.ReadAllText("C:\test.txt", _
   System.Text.Encoding.UTF32)
MsgBox(fileReader)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier n'existe pas (FileNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur E/S se produit (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- Il n'y a pas assez de mémoire pour écrire la chaîne dans la mémoire tampon (OutOfMemoryException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut que le fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Vérifiez toutes les entrées avant d'utiliser les données dans votre application. Le fichier n'a peut-être pas le contenu attendu, et les méthodes utilisées pour lire le fichier peuvent échouer.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Dépannage : lecture et écriture dans des fichiers texte

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.ReadAllText, méthode

Concepts

Codages de fichiers

Autres ressources

Lecture à partir de fichiers en Visual Basic

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

L'objet **TextFieldParser** permet d'analyser facilement et efficacement les fichiers texte structurés, tels que les journaux. La propriété **TextFieldType** définit s'il s'agit d'un fichier délimité ou d'un fichier dont les champs de texte ont une largeur fixe.

Pour analyser un fichier texte séparé par des virgules

1. Créez un **TextFieldParser**. Le code suivant crée le **TextFieldParser** nommé `MyReader` et ouvre le fichier `test.txt`.

   VB

   ```
   Using MyReader As New _
   Microsoft.VisualBasic.FileIO.TextFieldParser _
   ("C:\TestFolder\test.txt")
   ```

2. Définissez le type et le séparateur de **TextField**. Le code suivant définit la propriété **TextFieldType** comme `Delimited` et « , » comme séparateur.

   VB

   ```
   MyReader.TextFieldType = FileIO.FieldType.Delimited
   MyReader.SetDelimiters(",")
   ```

3. Parcourez les champs du fichier. Si des lignes sont endommagées, signalez une erreur poursuivez l'analyse. Le code suivant parcourt le fichier et affiche tour à tour chaque champ en signalant ceux qui sont mis en forme incorrectement.

   VB

   ```
   Dim currentRow As String()
      While Not MyReader.EndOfData
         Try
            currentRow = MyReader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
               MsgBox(currentField)
            Next
            Catch ex As _
            Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            "is not valid and will be skipped.")
        End Try
   ```

4. Fermez les blocs **While** et **Using** avec **End While** et **End Using**.

   VB

   ```
      End While
   End Using
   ```

Exemple

Cet exemple lit le fichier `test.txt`.

VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
   MyReader.TextFieldType = FileIO.FieldType.Delimited
   MyReader.SetDelimiters(",")
   Dim currentRow As String()
   While Not MyReader.EndOfData
      Try
         currentRow = MyReader.ReadFields()
         Dim currentField As String
         For Each currentField In currentRow
            MsgBox(currentField)
         Next
      Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
      MsgBox("Line " & ex.Message & _
      "is not valid and will be skipped.")
      End Try
   End While
End Using
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Une ligne ne peut pas être analysée avec le format spécifié (MalformedLineException). Le message d'exception spécifie la ligne qui provoque l'exception, tandis que le texte contenu dans la ligne est assigné à la propriété TextFieldParser.ErrorLine.
- Le fichier spécifié n'existe pas (FileNotFoundException).
- Une situation de niveau de confiance partiel dans laquelle l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier. (SecurityException).
- Le chemin d'accès est trop long (PathTooLongException
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic
Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Dépannage des exceptions : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

Référence

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

L'objet **TextFieldParser** permet d'analyser facilement et efficacement les fichiers texte structurés, tels que les journaux.

La propriété **TextFieldType** définit s'il s'agit d'un fichier délimité ou d'un fichier dont les champs de texte ont une largeur fixe. Pour spécifier un champ à largeur variable dans un fichier à largeur fixe, affectez à la largeur du champ la valeur -1.

Pour lire un fichier texte à largeur fixe

1. Créez un **TextFieldParser**. Le code suivant crée le **TextFieldParser** nommé `Reader` et ouvre le fichier `test.log`.

   VB

   ```
   Using Reader As New _
   Microsoft.VisualBasic.FileIO.TextFieldParser _
   ("C:\TestFolder\test.log")
   ```

2. Affectez à la propriété **TextFieldType** la valeur **FixedWidth** en définissant la largeur et le format. Le code suivant définit les colonnes de texte. La première a une largeur de 5 caractères, la deuxième de 10, la troisième de 11, tandis que la quatrième a une largeur variable.

   VB

   ```
   Reader.TextFieldType = _
   Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
   Reader.SetFieldWidths(5, 10, 11, -1)
   ```

3. Parcourez les champs du fichier. Si des lignes sont endommagées, signalez une erreur poursuivez l'analyse.

   VB

   ```
   Dim currentRow As String()
      While Not Reader.EndOfData
         Try
            currentRow = Reader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
               MsgBox(currentField)
            Next
         Catch ex As _
         Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            "is not valid and will be skipped.")
    End Try
   ```

4. Fermez les blocs **While** et **Using** avec **End While** et **End Using**.

   VB

   ```
      End While
   End Using
   ```

Exemple

Cet exemple lit le fichier `test.log`.

VB

```
Using Reader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\TestFolder\test.log")
   Reader.TextFieldType = _
   Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
   Reader.SetFieldWidths(5, 10, 11, -1)
   Dim currentRow As String()
   While Not Reader.EndOfData
      Try
         currentRow = Reader.ReadFields()
         Dim currentField As String
         For Each currentField In currentRow
            MsgBox(currentField)
         Next
      Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
         MsgBox("Line " & ex.Message & _
         "is not valid and will be skipped.")
      End Try
   End While
End Using
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Une ligne ne peut pas être analysée avec le format spécifié (MalformedLineException). Le message d'exception spécifie la ligne qui provoque l'exception, tandis que la propriété TextFieldParser.ErrorLine est assignée au texte contenu dans la ligne.
- Le fichier spécifié n'existe pas (FileNotFoundException).
- Une situation de niveau de confiance partiel dans laquelle l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier. (SecurityException).
- Le chemin d'accès est trop long (PathTooLongException
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic
Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Dépannage des exceptions : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

Référence

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

L'objet **TextFieldParser** permet d'analyser facilement et efficacement les fichiers texte structurés, tels que les journaux. Vous pouvez traiter un fichier à plusieurs formats à l'aide de la méthode **PeekChars** pour déterminer le format de chaque ligne à mesure que vous analysez le fichier.

Pour analyser un fichier texte avec plusieurs formats

1. Définissez le format attendu et le format utilisé lorsqu'une erreur est signalée.

   VB

   ```
   Dim StdFormat As Integer()= {5,10,11,-1}
   Dim ErrorFormat As Integer() = {5,5,-1}
   ```

2. Créez un objet **TextFieldParser** en définissant la largeur et le format.

   VB

   ```
   Using MyReader As New _
   Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
   MyReader.TextFieldType = FileIO.FieldType.FixedWidth
   ```

3. Parcourez les lignes en testant le format avant la lecture.

   VB

   ```
   Dim CurrentRow As String()
   While Not MyReader.EndOfData
      Try
         Dim RowType As String = MyReader.PeekChars(3)
         If String.Compare(RowType, "Err") = 0 Then
            ' If this line describes an error, the format of
            ' the row will be different.
            MyReader.SetFieldWidths(ErrorFormat)
            CurrentRow = MyReader.ReadFields
            MyReader.SetFieldWidths(StdFormat)
                      Else
            'Otherwise parse the fields normally
            CurrentRow = MyReader.ReadFields
            For Each newString As String In CurrentRow
               My.Computer.FileSystem.WriteAllText _
               ("newFile.txt", newString, True)
             Next
      End If
   ```

4. Écrivez les erreurs dans la console.

   VB

   ```
         Catch ex As _
         Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.")
         End Try
      End While
   ```

```
End Using
```

Exemple

Cet exemple lit le fichier `testfile.txt`.

VB

```
Dim StdFormat As Integer() = {5, 10, 11, -1}
Dim ErrorFormat As Integer() = {5, 5, -1}
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
   MyReader.TextFieldType = FileIO.FieldType.FixedWidth
   MyReader.FieldWidths = StdFormat
   Dim CurrentRow As String()
      While Not MyReader.EndOfData
         Try
            Dim RowType As String = MyReader.PeekChars(3)
            If String.Compare(RowType, "Err") = 0 Then
               ' If this line describes an error, the format of the row will be different.
               MyReader.SetFieldWidths(ErrorFormat)
               CurrentRow = MyReader.ReadFields
               MyReader.SetFieldWidths(StdFormat)
            Else
               ' Otherwise parse the fields normally
               CurrentRow = MyReader.ReadFields
               For Each newString As String In CurrentRow
                  My.Computer.FileSystem.WriteAllText("newFile.txt", newString, True)
               Next
            End If
         Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
         End Try
      End While
End Using
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Une ligne ne peut pas être analysée avec le format spécifié (MalformedLineException). Le message d'exception spécifie la ligne qui provoque l'exception, tandis que la propriété TextFieldParser.ErrorLine est assignée au texte contenu dans la ligne.
- Le fichier spécifié n'existe pas (FileNotFoundException).
- Une situation de niveau de confiance partiel dans laquelle l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier. (SecurityException).
- Le chemin d'accès est trop long (PathTooLongException
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Référence

TextFieldParser, objet
TextFieldParser.PeekChars, méthode
My.Computer.FileSystem.WriteAllText, méthode
TextFieldParser.EndOfData, propriété
TextFieldParser.TextFieldType, propriété
MalformedLineException

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Comment : lire des fichiers binaires dans Visual Basic

L'objet **My.Computer.FileSystem** fournit la méthode **ReadAllBytes** pour la lecture de fichiers binaires.

Pour lire un fichier binaire

- Utilisez la méthode **ReadAllBytes** qui retourne le contenu d'un fichier sous forme de tableau d'octets. Cet exemple lit le fichier `C:/Documents and Settings/selfportrait.jpg`.

  VB

```
My.Computer.FileSystem.ReadAllBytes _
("C:/Documents and Settings/selfportrait.jpg")
```

Programmation fiable

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier n'existe pas (FileNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur E/S se produit (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- Il n'y a pas assez de mémoire pour écrire la chaîne dans la mémoire tampon (OutOfMemoryException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut qu'un fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Vérifiez toutes les entrées avant d'utiliser les données dans votre application. Le fichier n'a peut-être pas le contenu attendu, et les méthodes utilisées pour lire le fichier peuvent échouer.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem.ReadAllBytes, méthode

My.Computer.FileSystem.WriteAllBytes, méthode

Autres ressources

Lecture à partir de fichiers en Visual Basic

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

Comment : lire des fichiers texte existants dans Mes documents (Visual Basic)

L'exemple de code suivant lit le contenu des fichiers texte du dossier **Mes documents** dans un fichier unique.

Exemple

VB

```
Dim filePaths As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
Dim allText As String
Try
   filePaths = My.Computer.FileSystem.GetFiles _
   (My.Computer.FileSystem.SpecialDirectories.MyDocuments)
   For Each file As String In filePaths
      allText = My.Computer.FileSystem.ReadAllText(file)
      My.Computer.FileSystem.WriteAllText("bigfile.txt", allText, True)
   Next
Catch fileException As Exception
   Throw fileException
End Try
```

Compilation du code

Remplacez `"bigfile.txt"` par le nom du fichier dans lequel vous souhaitez écrire.

Programmation fiable

Les fichiers lus doivent être des fichiers texte.

Vous pouvez utiliser le OpenFileDialog, composant (Windows Forms) et le SaveFileDialog, composant (Windows Forms) pour réduire la probabilité d'erreurs d'exécution relatives aux autorisations.

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut que le fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Vérifiez toutes les entrées avant d'utiliser les données dans votre application. Le fichier n'a peut-être pas le contenu attendu, et les méthodes utilisées pour lire le fichier peuvent échouer.

Sécurité

Pour lire un fichier, votre assembly requiert un niveau de privilège accordé par la classe FileIOPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code. L'utilisateur doit également avoir accès au fichier. Pour plus d'informations, consultez Listes de contrôle d'accès.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet
My.Computer.FileSystem.ReadAllText, méthode
My.Computer.FileSystem.WriteAllText, méthode
OpenFileDialog
SaveFileDialog

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

L'objet **My.Computer.FileSystem** fournit des méthodes pour ouvrir un TextReader et un TextWriter. Ces méthodes, **OpenTextFileWriter** et **OpenTextFileReader**, sont des méthodes avancées qui n'apparaissent pas dans IntelliSense tant que vous ne sélectionnez pas l'onglet **Tout**.

Pour lire une ligne d'un fichier à l'aide d'un lecteur de texte

- Utilisez la méthode **OpenTextFileReader** pour ouvrir le **TextReader** en spécifiant le fichier. Cet exemple ouvre le fichier nommé `testfile.txt`, en lit une ligne et affiche cette dernière dans un message.

  VB

```
Dim fileReader As System.IO.StreamReader
fileReader = _
My.Computer.FileSystem.OpenTextFileReader("C:\\testfile.txt")
Dim stringReader As String
stringReader = fileReader.ReadLine()
MsgBox("The first line of the file is " & stringReader)
```

Programmation fiable

Le fichier lu doit être un fichier texte.

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut qu'un fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Vérifiez toutes les entrées avant d'utiliser les données dans votre application. Le fichier n'a peut-être pas le contenu attendu, et les méthodes utilisées pour lire le fichier peuvent échouer.

Sécurité

Pour lire un fichier, votre assembly requiert un niveau de privilège accordé par la classe FileIOPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code. L'utilisateur doit également avoir accès au fichier. Pour plus d'informations, consultez Listes de contrôle d'accès.

Voir aussi

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.OpenTextFileWriter, méthode
My.Computer.FileSystem.OpenTextFileReader, méthode
OpenFileDialog

Autres ressources

SaveFileDialog, composant (Windows Forms)
Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

Cette section répertorie les tâches relatives à l'écriture dans les fichiers.

Dans cette section

Comment : insérer du texte dans des fichiers dans Visual Basic

Montre comment écrire dans des fichiers texte.

Comment : effectuer un ajout à des fichiers texte dans Visual Basic

Montre comment ajouter du texte à un fichier texte.

Comment : écrire dans des fichiers binaires dans Visual Basic

Montre comment écrire dans un fichier binaire.

Comment : insérer du texte dans les fichiers du répertoire Mes Documents dans Visual Basic

Montre comment créer et écrire dans un nouveau fichier texte du répertoire **Mes documents**.

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Montre comment écrire dans un fichier à l'aide d'un objet System.IO.StreamWriter.

Référence

My.Computer.FileSystem, objet

Décrit l'objet **My.Computer.FileSystem**, ses méthodes et ses propriétés.

My.Computer.FileSystem.OpenTextFileWriter, méthode

Décrit la méthode **OpenTextFileWriter**.

My.Computer.FileSystem.WriteAllBytes, méthode

Décrit la méthode **WriteAllBytes**.

My.Computer.FileSystem.WriteAllText, méthode

Décrit la méthode **WriteAllText**.

Rubriques connexes

Lecture à partir de fichiers en Visual Basic

Répertorie les tâches relatives la lecture de fichiers.

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Répertorie les tâches relatives à la création, à la suppression, au déplacement et à l'attribution de nouveaux noms aux fichiers et aux répertoires.

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Répertorie les tâches relatives à la définition des propriétés de fichier, de répertoire et de lecteur.

Comment : écrire dans le Presse-papiers en Visual Basic

Explique comment écrire des données dans le Presse-papiers.

Codages de fichiers

Fournit une vue d'ensemble des codages de fichier.

Comment : insérer du texte dans des fichiers dans Visual Basic

La méthode My.Computer.FileSystem.WriteAllText peut être utilisée pour écrire du texte dans les fichiers. Si le fichier cible n'existe pas, il est créé.

Procédure

Pour écrire du texte dans un fichier

- Utilisez la méthode **WriteAllText** pour écrire du texte dans un fichier en spécifiant le fichier et le texte à écrire. Cet exemple écrit la ligne `"This is new text."` dans le fichier nommé `test.txt` en ajoutant le texte à un texte existant dans le fichier.

  VB

```
My.Computer.FileSystem.WriteAllText("C:\TestFolder1\test.txt", _
"This is new text to be added.",True)
```

Pour écrire une série de chaînes dans un fichier

- Parcourez la collection de chaînes. Utilisez la méthode **WriteAllText** pour écrire du texte dans un fichier en spécifiant le fichier cible et la chaîne à ajouter et en affectant à *append* la valeur **True**.

  Cet exemple écrit les noms des fichiers du répertoire `Documents and Settings` dans `FileList.txt`, ce qui insère un retour chariot entre chacun d'eux pour une meilleure lisibilité.

  VB

```
For Each foundFile As String In _
My.Computer.FileSystem.GetFiles("C:\Documents and Settings")
foundFile = foundFile & vbCrLf
My.Computer.FileSystem.WriteAllText _
("C:\Documents and Settings\FileList.txt", foundFile, True)
Next
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *File* pointe vers un chemin d'accès qui n'existe pas (FileNotFoundException ou DirectoryNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur E/S se produit (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- Le disque est plein et l'appel à **WriteAllText** échoue (**IOException**).

Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Voir aussi

Tâches

Comment : lire des fichiers texte dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.WriteAllText, méthode

Comment : effectuer un ajout à des fichiers texte dans Visual Basic

Vous pouvez utiliser la méthode My.Computer.FileSystem.WriteAllText pour effectuer un ajout à un fichier texte en affectant la valeur **True** au paramètre *append*.

Pour effectuer un ajout à un fichier texte

- Utilisez la méthode **WriteAllText** en spécifiant le fichier cible et la chaîne à ajouter et en affectant au paramètre *append* la valeur **True**.

  Cet exemple écrit la chaîne `"This is a test string."` dans le fichier nommé `Testfile.txt`.

  VB

```
Dim inputString As String = "This is a test string."
My.Computer.FileSystem.WriteAllText _
("C://testfile.txt", inputString, True)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *File* pointe vers un chemin d'accès qui n'existe pas (FileNotFoundException ou DirectoryNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur E/S se produit (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Voir aussi

Référence

My.Computer.FileSystem.WriteAllText, méthode
My.Computer.FileSystem, objet

Autres ressources

Écriture dans des fichiers en Visual Basic

Comment : écrire dans des fichiers binaires dans Visual Basic

La méthode My.Computer.FileSystem.WriteAllBytes écrit des données dans un fichier binaire. Si le paramètre *append* a la valeur **True**, il ajoute les données au fichier ; sinon, les données du fichier sont remplacées.

Si le chemin d'accès spécifié excluant le nom de fichier n'est pas valide, une exception DirectoryNotFoundException est levée. Si le chemin d'accès est valide et que le fichier n'existe pas, le fichier est créé.

Pour écrire dans un fichier binaire

- Utilisez la méthode **WriteAllBytes**, en fournissant le chemin d'accès, le nom et les octets à écrire. Cet exemple ajoute le tableau de données `CustomerData` au fichier nommé `CollectedData.dat`.

  VB

```
My.Computer.FileSystem.WriteAllBytes _
("C:\MyDocuments\CustomerData", CustomerData, True)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception.

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides. (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *File* pointe vers un chemin d'accès qui n'existe pas (FileNotFoundException ou **DirectoryNotFoundException**).
- Le fichier est utilisé par un autre processus, ou une erreur E/S se produit (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Voir aussi

Tâches

Comment : insérer du texte dans des fichiers dans Visual Basic

Référence

My.Computer.FileSystem.WriteAllBytes, méthode

Comment : insérer du texte dans les fichiers du répertoire Mes Documents dans Visual Basic

L'objet **My.Computer.FileSystem.SpecialDirectories** vous permet d'accéder aux répertoires spéciaux, tels que le répertoire **Mes documents**.

Procédure

Pour écrire de nouveaux fichiers texte dans le répertoire Mes documents

1. Utilisez la propriété **My.Computer.FileSystem.SpecialDirectories.MyDocuments** pour fournir le chemin d'accès.

   VB
   ```
   Dim filePath As String
   filePath = System.IO.Path.Combine( _
   My.Computer.FileSystem.SpecialDirectories.MyDocuments, "test.txt")
   ```

2. Utilisez la méthode **WriteAllText** pour écrire du texte dans le fichier spécifié.

   VB
   ```
   My.Computer.FileSystem.WriteAllText(filePath, "some text", True)
   ```

Exemple

VB
```
Try
    Dim filePath As String
    filePath = System.IO.Path.Combine( _
               My.Computer.FileSystem.SpecialDirectories.MyDocuments, "test.txt")
           My.Computer.FileSystem.WriteAllText(filePath, "some text", False)
Catch fileException As Exception
    Throw fileException
End Try
```

Compilation du code

Remplacez `test.txt` par le nom du fichier dans lequel vous souhaitez écrire.

Programmation fiable

Ce code lève de nouveau toutes les exceptions qui peuvent se produire lorsque vous écrivez du texte dans le fichier. Vous pouvez réduire la probabilité d'exceptions en utilisant les contrôles Windows Forms, tels que les composants OpenFileDialog et SaveFileDialog qui limitent les choix de l'utilisateur aux noms de fichiers valides. Toutefois, l'utilisation de ces contrôles n'est pas sans poser de problèmes. Le système de fichiers peut changer entre le moment où l'utilisateur sélectionne un fichier et le moment où le code s'exécute. La gestion des exceptions est donc presque toujours nécessaire lors de l'utilisation de fichiers.

Sécurité

Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Cet exemple crée un fichier. Si une application doit créer un fichier, elle doit disposer de l'autorisation Créer pour le dossier. Les autorisations sont définies à l'aide de listes de contrôle d'accès. Si le fichier existe déjà, l'application a uniquement besoin de l'autorisation Écrire, ce qui représente un privilège inférieur. Lorsque cela est possible, il est plus sûr de créer le fichier au cours du déploiement et de n'accorder les privilèges Lire que sur un seul fichier (plutôt que les privilèges Créer sur un dossier). En outre, il est plus sûr d'écrire les données dans des dossiers utilisateur que dans le dossier racine ou le dossier **Program Files**. Pour plus d'informations, consultez Listes de contrôle d'accès.

Voir aussi

Référence

My.Computer, objet
My.Computer.FileSystem, objet
My.Computer.FileSystem.WriteAllText, méthode
My.Computer.FileSystem.SpecialDirectories, objet
System.IO.Path.Combine(System.String,System.String)

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Cet exemple ouvre un objet StreamWriter avec la méthode **My.Computer.FileSystem.OpenTextFileWriter** et l'utilise pour écrire une chaîne dans un fichier texte à l'aide de la méthode WriteLine de la classe **StreamWriter**.

Exemple

VB

```
Dim file As System.IO.StreamWriter
file = My.Computer.FileSystem.OpenTextFileWriter("c:\test.txt", True)
file.WriteLine("Here is the first string.")
file.Close()
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le fichier existe déjà et est en lecture seule (IOException).
- Le disque est saturé (**IOException**).
- Le nom de chemin d'accès est trop long (PathTooLongException

Sécurité

Cet exemple crée un fichier s'il n'existe pas. Si une application doit créer un fichier, elle doit disposer de l'accès **Create** pour le dossier. Si le fichier existe déjà, l'application n'a besoin que d'un accès **Write**, un privilège moins important. Dans la mesure du possible, et pour des raisons de sécurité, il est préférable de créer le fichier au moment du déploiement et d'accorder uniquement un accès **Read** à un seul fichier plutôt qu'un accès **Create** à l'ensemble du dossier.

Voir aussi

Tâches

Comment : lire des fichiers texte dans Visual Basic

Référence

My.Computer.FileSystem.OpenTextFileWriter, méthode

StreamWriter

Autres ressources

Écriture dans des fichiers en Visual Basic

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Cette section répertorie les tâches associées à la création, la suppression, le déplacement et l'attribution de nouveaux noms aux fichiers et aux répertoires en Visual Basic.

Dans cette section

Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic

Montre comment copier des fichiers dotés d'un modèle de nom spécifique, par exemple des fichiers .txt, dans un répertoire.

Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic

Montre comment créer une copie d'un fichier dans le même répertoire.

Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic

Montre comment copier un fichier dans un autre répertoire.

Comment : créer un fichier dans Visual Basic

Montre comment créer un fichier.

Comment : supprimer un fichier dans Visual Basic

Montre comment supprimer un fichier.

Comment : supprimer tous les fichiers d'un répertoire dans Visual Basic

Montre comment supprimer tous les fichiers d'un répertoire spécifique.

Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic

Montre comment répertorier uniquement les fichiers dotés d'un modèle de nom spécifique dans un répertoire.

Comment : déplacer un fichier dans Visual Basic

Montre comment déplacer un fichier dans un autre répertoire.

Comment : déplacer une collection de fichiers dans Visual Basic

Montre comment déplacer de nombreux fichiers dans un autre répertoire.

Comment : renommer un fichier dans Visual Basic

Montre comment renommer un fichier.

Comment : renommer un répertoire dans Visual Basic

Montre comment renommer un répertoire.

Comment : copier un répertoire dans un autre répertoire dans Visual Basic

Montre comment copier un répertoire dans un autre emplacement.

Comment : créer un répertoire dans Visual Basic

Montre comment créer un répertoire.

Comment : supprimer un répertoire dans Visual Basic

Montre comment supprimer un répertoire.

Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic

Montre comment répertorier des répertoires dotés d'un modèle de nom spécifique.

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic

Montre comment répertorier les fichiers d'un répertoire.

Comment : obtenir le nombre de fichiers d'un répertoire dans Visual Basic

Montre comment déterminer le nombre de fichiers d'un répertoire.

Comment : déplacer un répertoire dans Visual Basic

Montre comment déplacer un répertoire.

Comment : déplacer le contenu d'un répertoire dans Visual Basic

Montre comment déplacer le contenu d'un répertoire.

Comment : récupérer le contenu du répertoire Mes Documents dans Visual Basic

Montre comment lire des répertoires spéciaux.

Comment : analyser des chemins d'accès dans Visual Basic

Montre comment utiliser les méthodes **My** pour combiner des chemins d'accès de fichier.

Référence

My.Computer.FileSystem, objet

Décrit l'objet **My.Computer.FileSystem** et ses membres.

My.Computer.FileSystem.CombinePath, méthode

Décrit la méthode **CombinePath**.

My.Computer.FileSystem.CopyDirectory, méthode

Décrit la méthode **CopyDirectory**.

My.Computer.FileSystem.CopyFile, méthode

Décrit la méthode **CopyFile**.

My.Computer.FileSystem.CreateDirectory, méthode

Décrit la méthode **CreateDirectory**.

My.Computer.FileSystem.DeleteDirectory, méthode

Décrit la méthode **DeleteDirectory**.

My.Computer.FileSystem.DeleteFile, méthode

Décrit la méthode **DeleteFile**.

My.Computer.FileSystem.GetParentPath, méthode

Décrit la méthode **GetParentPath**.

My.Computer.FileSystem.MoveDirectory, méthode

Décrit la méthode **MoveDirectory**.

My.Computer.FileSystem.MoveFile, méthode

Décrit la méthode **MoveFile**.

My.Computer.FileSystem.RenameDirectory, méthode

Décrit la méthode **RenameDirectory**.

My.Computer.FileSystem.RenameFile, méthode

Décrit la méthode **RenameFile**.

My.Computer.FileSystem.SpecialDirectories, objet

Décrit l'objet **SpecialDirectories**.

Rubriques connexes

Lecture à partir de fichiers en Visual Basic

Répertorie les tâches associées à la lecture de fichiers.

Écriture dans des fichiers en Visual Basic

Répertorie les tâches relatives à l'écriture dans les fichiers.

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Répertorie les tâches associées à l'obtention et à la définition des propriétés de fichier, de répertoire et de lecteur.

Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic

La méthode My.Computer.FileSystem.GetFiles retourne une collection en lecture seule des chaînes représentant les noms de chemins d'accès des fichiers. Vous pouvez utiliser le paramètre *wildCards* pour spécifier un modèle spécifique.

Une collection vide est retournée si aucun fichier correspondant n'est trouvé.

Vous pouvez utiliser la méthode My.Computer.FileSystem.CopyFile pour copier des fichiers dans un répertoire.

Pour copier des fichiers d'un modèle spécifique dans un répertoire

1. Utilisez la méthode **GetFiles** pour retourner la liste de fichiers. Cet exemple retourne tous les fichiers .rtf dans le répertoire spécifié.

   VB

   ```
   For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
       My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
       FileIO.SearchOption.SearchTopLevelOnly, "*.rtf")
   ```

2. Utilisez la méthode **CopyFile** pour copier des fichiers. Cet exemple copie les fichiers dans le répertoire nommé `testdirectory`.

   VB

   ```
   My.Computer.FileSystem.CopyFile(foundFile, "C:\testdirectory\" & foundFile)
   ```

3. Fermez l'instruction **For** par une instruction **Next**.

   VB

   ```
   Next
   ```

Exemple

L'exemple suivant qui présente les extraits de code ci-dessus sous leur forme complète, copie tous les fichiers .rtf du répertoire spécifié dans le répertoire nommé `testdirectory`.

VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchTopLevelOnly, "*.rtf")

    My.Computer.FileSystem.CopyFile(foundFile, "C:\testdirectory\" & foundFile)
Next
```

Sécurité

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).

- Le répertoire n'existe pas (DirectoryNotFoundException).
- Le répertoire pointe vers un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException). L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Comment : placer la collection de fichiers dans un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.CopyFile, méthode
My.Computer.FileSystem.GetFiles, méthode

Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic

Utilisez la méthode **My.Computer.FileSystem.CopyFile** pour copier des fichiers. Les paramètres vous permettent de remplacer des fichiers existants, de renommer le fichier, d'afficher la progression de l'opération et d'autoriser l'utilisateur à annuler l'opération.

Pour créer une copie d'un fichier dans le même dossier

- Utilisez la méthode **CopyFile** en fournissant le fichier cible et l'emplacement. L'exemple suivant crée une copie de `test.txt` appelée `test2.txt`.

  VB

```
My.Computer.FileSystem.CopyFile("C:\TestFolder\test.txt", _
"C:\TestFolder\test2.txt", Microsoft.VisualBasic.FileIO.UIOption.OnlyErrorDialogs, Fil
eIO.UICancelOption.DoNothing)
```

Pour créer une copie d'un fichier dans le même dossier en remplaçant les fichiers existants

- Utilisez la méthode **CopyFile** en fournissant le fichier cible et l'emplacement et en affectant à *overwrite* la valeur **True**. L'exemple suivant crée une copie de `test.txt` appelée `test2.txt` et remplace tous les fichiers existants par ce nom.

  VB

```
My.Computer.FileSystem.CopyFile("C:\TestFolder\test.txt", _
"C:\TestFolder\test2.txt", True)
```

Programmation fiable

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le système n'a pas pu récupérer le chemin d'accès absolu (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Le chemin d'accès combiné pointe sur un répertoire existant (IOException).
- Le fichier de destination existe et *overwrite* a la valeur **False** (**IOException**).
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (**IOException**).
- Un fichier du dossier cible portant le même nom est en cours d'utilisation (**IOException**).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- *ShowUI* a la valeur **True**, *onUserCancel* a la valeur **ThrowException** et l'utilisateur a annulé l'opération (OperationCanceledException).
- *ShowUI* a la valeur **True**, *onUserCancel* a la valeur **ThrowException** et une erreur E/S non spécifiée se produit (**OperationCanceledException**).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).

- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Voir aussi

Tâches

Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic
Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic
Comment : copier un répertoire dans un autre répertoire dans Visual Basic
Comment : renommer un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.CopyFile, méthode
UICancelOption, énumération

Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic

La méthode **My.Computer.FileSystem.CopyFile** vous permet de copier des fichiers. Ses paramètres permettent de remplacer les fichiers existants, de renommer le fichier, d'afficher la progression de l'opération et d'autoriser l'utilisateur à annuler l'opération.

Pour copier un fichier texte dans un autre dossier

- Utilisez la méthode **CopyFile** pour copier un fichier en spécifiant un fichier source et le répertoire cible. Le paramètre *overwrite* vous permet de spécifier s'il faut ou non remplacer les fichiers existants. Les exemples de code suivants montrent comment utiliser **CopyFile**.

  VB

```
' Copy the file to a new location without overwriting existing file.
My.Computer.FileSystem.CopyFile( _
    "C:\UserFiles\TestFiles\testFile.txt", _
    "C:\UserFiles\TestFiles2\testFile.txt")

' Copy the file to a new folder, overwriting existing file.
My.Computer.FileSystem.CopyFile( _
    "C:\UserFiles\TestFiles\testFile.txt", _
    "C:\UserFiles\TestFiles2\testFile.txt", _
    FileIO.UIOption.AllDialogs, _
    FileIO.UICancelOption.DoNothing)

' Copy the file to a new folder and rename it.
My.Computer.FileSystem.CopyFile( _
    "C:\UserFiles\TestFiles\testFile.txt", _
    "C:\UserFiles\TestFiles2\NewFile.txt", _
    FileIO.UIOption.AllDialogs, _
    FileIO.UICancelOption.DoNothing)
```

Programmation fiable

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le système n'a pas pu récupérer le chemin d'accès absolu (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Le chemin d'accès combiné pointe sur un répertoire existant (IOException).
- Le fichier de destination existe et *overwrite* a la valeur **False** (**IOException**).
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (**IOException**).
- Un fichier du dossier cible portant le même nom est en cours d'utilisation (**IOException**).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- *ShowUI* a la valeur **True**, *onUserCancel* a la valeur **ThrowException** et l'utilisateur a annulé l'opération

(OperationCanceledException).

- *ShowUI* a la valeur **True**, *onUserCancel* a la valeur **ThrowException** et une erreur E/S non spécifiée se produit (**OperationCanceledException**).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Voir aussi

Tâches

Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic
Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic
Comment : copier un répertoire dans un autre répertoire dans Visual Basic
Comment : renommer un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.CopyFile, méthode
UICancelOption, énumération

Comment : créer un fichier dans Visual Basic

Cet exemple crée un fichier texte vide dans le chemin d'accès spécifié à l'aide de la méthode Create de la classe File.

Exemple

VB

```
Dim file As System.IO.FileStream
file = System.IO.File.Create("c:\test.txt")
```

Compilation du code

Utilisez la variable `file` pour écrire dans le fichier.

Programmation fiable

Si le fichier existe déjà, il est remplacé.

Les conditions ci-dessous peuvent générer une exception :

- Le nom du chemin d'accès est incorrect. Il contient par exemple des caractères non valides ou se compose uniquement d'un espace blanc (ArgumentException).
- Le chemin d'accès est en lecture seule (classe IOException).
- Le nom du chemin d'accès est **Nothing** (ArgumentNullException).
- Le nom du chemin d'accès est trop long (PathTooLongException).
- Le chemin d'accès n'est pas valide (DirectoryNotFoundException).
- Le chemin d'accès se compose uniquement du signe deux-points « : » (NotSupportedException).

Sécurité

Une SecurityException peut être levée dans des environnements où le niveau de confiance n'est pas total.

L'appel à la méthode **Create** requiert FileIOPermission.

Une exception UnauthorizedAccessException est levée si l'utilisateur n'a pas l'autorisation de créer le fichier.

Voir aussi

Référence

System.IO

Create

Concepts

Utilisation de bibliothèques à partir de code d'un niveau de confiance partiel

Notions fondamentales de la sécurité d'accès du code

Comment : supprimer un fichier dans Visual Basic

La méthode **DeleteFile** de l'objet **My.Computer.FileSystem** vous permet de supprimer un fichier. Elle offre entre autres les options suivantes : envoyer ou non le fichier supprimé à la **Corbeille**, demander ou non à l'utilisateur de confirmer que le fichier doit être supprimé et ce qu'il faut faire lorsque l'utilisateur annule l'opération.

Pour supprimer un fichier texte

- Utilisez la méthode **DeleteFile** pour supprimer le fichier. Le code suivant illustre comment supprimer le fichier nommé `test.txt`.

  VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt")
```

Pour supprimer un fichier texte et demander à l'utilisateur de confirmer que le fichier doit être supprimé

- Utilisez la méthode **DeleteFile** pour supprimer le fichier, en affectant *showUI* à **AllDialogs**. Le code suivant illustre comment supprimer le fichier nommé `test.txt` et permettre à l'utilisateur de confirmer que le fichier doit être supprimé.

  VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt", _
        FileIO.UIOption.AllDialogs, FileIO.RecycleOption.DeletePermanently, FileIO.UIC
ancelOption.DoNothing)
```

Pour supprimer un fichier texte et l'envoyer à la Corbeille

- Utilisez la méthode **DeleteFile** pour supprimer le fichier, en spécifiant **SendToRecycleBin** pour le paramètre *recycle*. Le code suivant illustre comment supprimer le fichier nommé `test.txt` et l'envoyer à la **Corbeille**.

  VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt", _
FileIO.UIOption.AllDialogs, FileIO.RecycleOption.SendToRecycleBin)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- Le fichier est en cours d'utilisation (IOException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- Le fichier n'existe pas (FileNotFoundException).
- L'utilisateur n'a pas l'autorisation nécessaire pour supprimer le fichier, ou le fichier est en lecture seule

(UnauthorizedAccessException).

- Il existe une situation de niveau de confiance partiel où l'utilisateur n'a pas les autorisations suffisantes (**SecurityException**).

- L'utilisateur a annulé l'opération et *onUserCancel* a la valeur Microsoft.VisualBasic.FileIO.UICancelOption.ThrowException (OperationCanceledException).

Voir aussi

Tâches

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Comment : supprimer un répertoire dans Visual Basic

Référence

UICancelOption, énumération
My.Computer.FileSystem, objet
UIOption, énumération
RecycleOption, énumération

Comment : supprimer tous les fichiers d'un répertoire dans Visual Basic

La méthode **DeleteFile** de l'objet **My.Computer.FileSystem** vous permet de supprimer un fichier. Elle offre entre autres les options suivantes : envoyer ou non le fichier supprimé à la **Corbeille**, demander ou non à l'utilisateur de confirmer que le fichier doit être supprimé et ce qu'il faut faire lorsque l'utilisateur annule l'opération.

Pour supprimer tous les fichiers d'un dossier

1. Utilisez la méthode **My.Computer.FileSystem.GetFiles** pour retourner la collection de chaînes représentant les fichiers du répertoire.
2. Utilisez une boucle **For...Each** avec la méthode **DeleteFile** pour supprimer tour à tour chaque fichier.

   L'exemple suivant supprime tous les fichiers du dossier **Mes documents**.

   VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.*")

    My.Computer.FileSystem.DeleteFile(foundFile, _
        FileIO.UIOption.AllDialogs, _
        FileIO.RecycleOption.DeletePermanently)
Next
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- Le fichier est en cours d'utilisation (IOException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- Le fichier n'existe pas (FileNotFoundException).
- L'utilisateur n'a pas l'autorisation nécessaire pour supprimer le fichier, ou le fichier est en lecture seule (UnauthorizedAccessException).
- Il existe une situation de niveau de confiance partiel où l'utilisateur n'a pas les autorisations suffisantes (**SecurityException**).
- L'utilisateur a annulé l'opération et *onUserCancel* a la valeur Microsoft.VisualBasic.FileIO.UICancelOption.ThrowException (OperationCanceledException).

Voir aussi

Tâches

Comment : supprimer un fichier dans Visual Basic
Comment : supprimer un répertoire dans Visual Basic

Comment : renommer un fichier dans Visual Basic
Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.DeleteFile, méthode
RecycleOption, énumération
UICancelOption, énumération

Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic

La méthode My.Computer.FileSystem.GetFiles retourne une collection en lecture seule des chaînes représentant les noms de chemins d'accès des fichiers. Vous pouvez utiliser le paramètre *wildCards* pour spécifier un modèle spécifique. Si vous souhaitez inclure des sous-répertoires dans la recherche, affectez le paramètre *searchType* à **SearchOption.SearchAllSubDirectories**.

Une collection vide est retournée si aucun fichier correspondant au modèle spécifié n'est trouvé.

Pour rechercher des fichiers avec un modèle spécifié

- Utilisez la méthode **GetFiles** en fournissant le nom et le chemin d'accès du répertoire que vous souhaitez rechercher et en spécifiant le modèle. L'exemple suivant retourne tous les fichiers avec l'extension `.dll` dans le répertoire et les ajoute à `ListBox1`.

  VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.dll")

    Listbox1.Items.Add(foundFile)
Next
```

Sécurité

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe sur un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Comment : placer la collection de fichiers dans un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.GetFiles, méthode

Comment : déplacer un fichier dans Visual Basic

La méthode **My.Computer.FileSystem.MoveFile** peut être utilisée pour déplacer un fichier dans un autre dossier. Si la structure cible n'existe pas, elle sera créée.

Pour déplacer un fichier

- Utilisez la méthode **MoveFile** pour déplacer le fichier, en spécifiant le nom et l'emplacement du fichier source et du fichier cible. Cet exemple déplace le fichier nommé `test.txt` de `TestDir1` vers `TestDir2`. Notez que le nom du fichier cible est spécifié même s'il est identique à celui du fichier source.

  VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", _
    "C:\TestDir2\test.txt")
```

Pour déplacer un fichier et le renommer

- Utilisez la méthode **MoveFile** pour déplacer le fichier, en spécifiant le nom et l'emplacement du fichier source, l'emplacement cible et le nouveau nom à l'emplacement cible. Cet exemple déplace le fichier nommé `test.txt` de `TestDir1` vers `TestDir2` et le renomme `nexttest.txt`.

  VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", _
    "C:\TestDir2\nexttest.txt", _
    FileIO.UIOption.AllDialogs, _
    FileIO.UICancelOption.ThrowException)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *destinationFileName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Le chemin d'accès combiné pointe sur un répertoire existant, le fichier de destination existe et *overwrite* a la valeur **False**, un fichier du répertoire cible portant le même nom est en cours d'utilisation, ou l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (IOException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- *showUI* a la valeur **True**, *onUserCancel* a la valeur **ThrowException** et l'utilisateur a annulé l'opération ou une erreur E/S non spécifiée se produit (OperationCanceledException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : renommer un fichier dans Visual Basic

Comment : déplacer une collection de fichiers dans Visual Basic

Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic

Comment : analyser des chemins d'accès dans Visual Basic

Comment : déplacer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.MoveFile, méthode

Comment : déplacer une collection de fichiers dans Visual Basic

La méthode **My.Computer.FileSystem.MoveFile** vous permet de déplacer des fichiers entre les répertoires.

Si la structure cible n'existe pas, elle sera créée.

Pour déplacer une collection de fichiers d'un répertoire à un autre

- Définissez la collection de fichiers et appelez la méthode **MoveFile**. Cet exemple déplace tous les fichiers du répertoire `MyDocuments` au dossier `StorageDir`.

  VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.*")

    Dim foundFileInfo As New System.IO.FileInfo(foundFile)
    My.Computer.FileSystem.MoveFile(foundFile, "C:\StorageDir\" & foundFileInfo.Name)
Next
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Le chemin d'accès combiné pointe sur un répertoire existant, le fichier de destination existe et *overwrite* a la valeur **False**, un fichier du répertoire cible portant le même nom est en cours d'utilisation, ou l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (IOException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- *showUI* a la valeur **True**, *onUserCancelOption* a la valeur **ThrowException** et l'utilisateur a annulé l'opération ou une erreur E/S non spécifiée se produit (OperationCanceledException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : renommer un fichier dans Visual Basic
Comment : déplacer un fichier dans Visual Basic
Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic
Comment : déplacer un répertoire dans Visual Basic
Comment : déplacer le contenu d'un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.MoveFile, méthode
My.Computer.FileSystem.GetFiles, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Comment : renommer un fichier dans Visual Basic

Utilisez la méthode **RenameFile** de l'objet **My.Computer.FileSystem** pour renommer un fichier en fournissant l'emplacement actuel, le nom de fichier et le nouveau nom de fichier. Cette méthode ne peut pas être utilisée pour déplacer un fichier ; utilisez plutôt la méthode **MoveFile** pour déplacer et renommer le fichier.

Pour renommer un fichier

- Utilisez la méthode **My.Computer.FileSystem.RenameFile** pour renommer un fichier. Cet exemple renomme le fichier `Test.txt` dans `SecondTest.txt`.

  VB

```
' Change "c:\test.txt" to the path and filename for the file that
' you want to rename.
My.Computer.FileSystem.RenameFile("C:\Test.txt", "SecondTest.txt")
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, l'extrait est localisé dans **Système de fichiers - Traitement de lecteurs, de dossiers et de fichiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- *newName* contient des informations relatives au chemin d'accès (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *newName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Un fichier ou répertoire existe avec le nom spécifié dans *newName* (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : déplacer un fichier dans Visual Basic
Comment : déplacer une collection de fichiers dans Visual Basic
Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic
Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic

Référence

My.Computer.FileSystem.RenameFile, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Comment : renommer un répertoire dans Visual Basic

Vous pouvez utiliser la méthode **RenameDirectory** de l'objet **My.Computer.FileSystem** pour renommer un répertoire en fournissant l'emplacement actuel et le nom du répertoire, ainsi que le nouveau nom du répertoire. Cette méthode ne peut pas être utilisée pour déplacer un répertoire ; utilisez plutôt la méthode **MoveDirectory** pour déplacer et renommer le répertoire.

Pour renommer un répertoire

- Utilisez la méthode **My.Computer.FileSystem.RenameDirectory** pour renommer un répertoire. L'exemple de code suivant remplace le nom du répertoire `Test` par `SecondTest`.

  VB

```
My.Computer.FileSystem.RenameDirectory("C:MyDocuments\Test", _
"SecondTest")
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, l'exemple est localisé dans **Système de fichiers - Traitement de lecteurs, de dossiers et de fichiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- *newName* contient des informations relatives au chemin d'accès (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *newName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le répertoire source n'est pas valide ou n'existe pas (DirectoryNotFoundException).
- Un fichier ou dossier existe avec le nom spécifié dans *newName* (IOException).
- Le dossier est un dossier racine (**IOException**).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : copier un répertoire dans un autre répertoire dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic
Comment : déplacer un répertoire dans Visual Basic
Comment : déplacer le contenu d'un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.RenameDirectory, méthode

Comment : copier un répertoire dans un autre répertoire dans Visual Basic

Utilisez la méthode My.Computer.FileSystem.CopyDirectory pour copier un répertoire dans un autre. Cette méthode copie le contenu du répertoire, ainsi que le répertoire lui-même. Si le répertoire cible n'existe pas, il sera créé. Si un répertoire portant le même nom existe à l'emplacement cible et que *overwrite* a la valeur **False**, le contenu des deux répertoires sera fusionné. Vous pouvez spécifier un nouveau nom pour le répertoire pendant l'opération.

Lors de la copie de fichiers dans un répertoire, des exceptions provoquées par un fichier spécifique, comme un fichier existant au cours d'une fusion alors que *overwrite* a la valeur **False**, peuvent être levées. Lorsque de telles exceptions sont levées, elles sont consolidées en une seule exception dont la propriété `Data` comporte des entrées dans lesquelles le chemin d'accès au fichier ou au répertoire constitue la clé et le message d'exception spécifique est contenu dans la valeur correspondante.

Pour copier un répertoire dans un autre répertoire

- Utilisez la méthode **CopyDirectory** en spécifiant les noms de répertoires source et de destination. L'exemple suivant copie le répertoire nommé `TestDirectory1` dans `TestDirectory2` en remplaçant les fichiers existants.

  VB

  ```
  My.Computer.FileSystem.CopyDirectory("C:\TestDirectory1", "C:\TestDirectory2", True)
  ```

  Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il est localisé dans **Système de fichiers - Traitement de lecteurs, de dossiers et de fichiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le nouveau nom spécifié pour le répertoire contient le signe deux-points (:) ou une barre oblique (\ ou /) (ArgumentException).
- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *destinationDirectoryName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le dossier source n'existe pas (DirectoryNotFoundException).
- La source est un répertoire racine (IOException).
- Le chemin d'accès combiné pointe sur un fichier existant (**IOException**).
- Le chemin source et le chemin cible sont identiques (**IOException**).
- *ShowUI* a la valeur **UIOption.AllDialogs** et l'utilisateur annule l'opération, ou un ou plusieurs fichiers du répertoire ne peuvent pas être copiés (OperationCanceledException).
- L'opération est cyclique (InvalidOperationException).
- Le chemin d'accès contient le signe deux-points (:) (NotSupportedException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (**NotSupportedException**).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

- Un fichier de destination existe mais est inaccessible (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic
Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Comment : déplacer un répertoire dans Visual Basic
Comment : déplacer le contenu d'un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.CopyDirectory, méthode

Comment : créer un répertoire dans Visual Basic

Utilisez la méthode **CreateDirectory** de l'objet **My.Computer.FileSystem** pour créer des répertoires.

Si le répertoire existe déjà, aucune exception n'est levée.

Pour créer un répertoire

- Utilisez la méthode **CreateDirectory** en spécifiant le chemin d'accès complet de l'emplacement où le répertoire doit être créé. Cet exemple crée le répertoire `NewDirectory` dans `C:\Documents and Settings\All Users\Documents`.

  VB

  ```
  My.Computer.FileSystem.CreateDirectory _
  ("C:\Documents and Settings\All Users\Documents\NewDirectory")
  ```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès est incorrect. Il contient par exemple des caractères non valides ou se compose uniquement d'un espace blanc (ArgumentException).
- Le répertoire parent du répertoire à créer est en lecture seule (IOException).
- Le chemin d'accès est **Nothing** (ArgumentNullException).
- Le nom de répertoire est trop long (PathTooLongException).
- Le nom de répertoire est formé du signe deux points « : » (NotSupportedException).
- L'utilisateur n'est pas autorisé à créer le répertoire (UnauthorizedAccessException).
- L'utilisateur n'a pas les autorisations nécessaires dans une situation de niveau de confiance partiel (SecurityException).

Voir aussi

Tâches

Comment : déterminer si un répertoire existe dans Visual Basic

Référence

My.Computer.FileSystem.CreateDirectory, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Comment : supprimer un répertoire dans Visual Basic

Utilisez la méthode **DeleteDirectory** de l'objet **My.Computer.FileSystem** pour supprimer un répertoire. Les options sont, entre autres, les suivantes : s'il faut ou non supprimer le contenu du répertoire, s'il faut ou non envoyer le répertoire supprimé à la **Corbeille** et s'il faut ou non afficher la progression de la suppression.

Pour supprimer un répertoire uniquement s'il est vide

- Utilisez la méthode **DeleteDirectory** pour supprimer le répertoire en spécifiant **False** pour *onDirectoryNotEmpty*. Cet exemple supprime le répertoire nommé `OldDirectory` uniquement s'il est vide.

  VB

```
My.Computer.FileSystem.DeleteDirectory("C:\OldDirectory", _
FileIO.DeleteDirectoryOption.ThrowIfDirectoryNonEmpty)
```

Pour supprimer un répertoire et l'envoyer à la Corbeille

- Utilisez la méthode **DeleteDirectory** pour supprimer le répertoire en spécifiant **RecycleOption.SendToRecycleBin** pour *recycle*. Cet exemple supprime le répertoire nommé `OldDirectory` et tout son contenu, en les envoyant à la **Corbeille** et en affichant la progression de l'opération.

  VB

```
My.Computer.FileSystem.DeleteDirectory("C:\OldDirectory", FileIO.UIOption.AllDialogs,
FileIO.RecycleOption.SendToRecycleBin)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès est une chaîne de longueur nulle, est incorrect, se compose uniquement d'un espace blanc ou contient des caractères incorrects (y compris des caractères génériques) (ArgumentException).
- Le chemin d'accès est un chemin de périphérique (commence par \\. \) (**ArgumentException**).
- Le chemin d'accès est **Nothing** (ArgumentNullException).
- Le répertoire n'existe pas ou est un fichier (DirectoryNotFoundException).
- L'utilisateur n'a pas l'autorisation de supprimer le répertoire ou le sous-répertoire (IOException).
- Un fichier du répertoire ou du sous-répertoire est en cours d'utilisation (**IOException**).
- Un nom de fichier ou de répertoire contient le signe deux-points (:) (NotSupportedException).
- *OnUserCancel* a la valeur **ThrowException** et l'utilisateur annule l'opération (OperationCanceledException).
- *OnUserCancel* a la valeur **ThrowException** et le répertoire ne peut pas être supprimé (**OperationCanceledException**).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- *showUI* a la valeur **AllDialogs** et l'utilisateur n'a pas les autorisations requises (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : supprimer un fichier dans Visual Basic

Référence

UICancelOption, énumération

My.Computer.FileSystem.DeleteDirectory, méthode

RecycleOption, énumération

UIOption, énumération
DeleteDirectoryOption, énumération

Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic

La méthode My.Computer.FileSystem.GetDirectories retourne une collection en lecture seule des chaînes représentant les noms de chemins d'accès des sous-répertoires d'un répertoire donné. Vous pouvez utiliser le paramètre *wildCards* pour spécifier un modèle spécifique. Si vous souhaitez inclure le contenu des sous-répertoires dans la recherche, affectez le paramètre *searchType* à **SearchOption.SearchAllSubDirectories**.

Une collection vide est retournée si aucun répertoire correspondant au modèle spécifié n'est trouvé.

Pour rechercher des sous-répertoires d'un modèle spécifique

- Utilisez la méthode **GetDirectories** en fournissant le nom et le chemin d'accès du répertoire à rechercher. L'exemple suivant retourne tous les répertoires de la structure de répertoire qui contiennent le mot « Logs » dans leur nom et les ajoute à `ListBox1`.

  VB

```
For Each foundDirectory As String In _
  My.Computer.FileSystem.GetDirectories( _
  My.Computer.FileSystem.SpecialDirectories.MyDocuments, True, _
  "*Logs*")

  ListBox1.Items.Add(foundDirectory)
Next
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Un ou plusieurs des caractères génériques spécifiés ont une valeur **Nothing**, constituent une chaîne vide ou contiennent uniquement des espaces (**ArgumentNullException**).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe sur un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic

Référence

My.Computer.FileSystem.GetDirectories, méthode

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic

La méthode My.Computer.FileSystem.GetFiles retourne une collection en lecture seule de chaînes qui représentent le nom des fichiers d'un répertoire. Vous pouvez utiliser le paramètre *wildCards* pour spécifier un modèle spécifique. Pour inclure des sous-répertoires dans la recherche, affectez le paramètre *searchType* à **SearchOption.SearchAllSubDirectories**.

Une collection vide est retournée si aucun fichier correspondant au modèle spécifié n'est trouvé.

Pour répertorier les fichiers d'un répertoire

- Utilisez la méthode **GetFiles** en fournissant le nom et le chemin d'accès du répertoire à rechercher. L'exemple suivant retourne tous les fichiers dans le répertoire et les ajoute à `ListBox1`.

  VB

  ```
  For Each foundFile As String In My.Computer.FileSystem.GetFiles _
  (My.Computer.FileSystem.SpecialDirectories.MyDocuments)
      listBox1.Items.Add(foundFile)
  Next
  ```

  Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il est localisé dans **Système de fichiers - Traitement de lecteurs, de dossiers et de fichiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe sur un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic
Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic

Référence

My.Computer.FileSystem.GetFiles, méthode

Comment : obtenir le nombre de fichiers d'un répertoire dans Visual Basic

Vous pouvez utiliser la méthode My.Computer.FileSystem.GetFiles pour retourner une collection en lecture seule de chaînes représentant les noms des fichiers dans le répertoire spécifié. Ensuite, vous pouvez utiliser la propriété Count pour déterminer le nombre de fichiers.

Pour déterminer le nombre de fichiers dans un répertoire

1. Utilisez la méthode **GetFiles** pour retourner la collection de fichiers dans le répertoire spécifié. Cet exemple retourne les fichiers dans le répertoire nommé `TestDir`.

   VB
   ```
   Dim counter As _
   System.Collections.ObjectModel.ReadOnlyCollection(Of String)
   counter = My.Computer.FileSystem.GetFiles("C:\TestDir")
   ```

2. Utilisez la propriété **Count** pour déterminer le nombre de fichiers dans la collection. Cet exemple affiche le résultat dans une boîte de message.

   VB
   ```
   MsgBox("number of files is " & CStr(counter.Count))
   ```

Exemple

Cet exemple, qui présente l'extrait de code ci-dessus dans un formulaire complet, compte le nombre de fichiers dans `TestDir` et le signale dans une boîte de message.

VB
```
Dim counter As _
System.Collections.ObjectModel.ReadOnlyCollection(Of String)
counter = My.Computer.FileSystem.GetFiles("C:\TestDir")
MsgBox("number of files is " & CStr(counter.Count))
```

Compilation du code

Cet exemple nécessite les actions ou les éléments suivants :

- accéder aux membres de l'espace de noms System.Collections ; ajouter une instruction **Imports** si vous n'avez pas de noms de membres qualifiés complets dans votre code. Pour plus d'informations, consultez Imports, instruction ;
- un répertoire nommé `TestDir` à l'emplacement spécifié. Remplacez le chemin d'accès par le chemin d'accès du répertoire que vous souhaitez examiner.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le répertoire n'existe pas ou est un fichier (DirectoryNotFoundException).
- *directory* pointe sur un fichier existant (IOException).

- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic

Référence

My.Computer.FileSystem.GetFiles, méthode

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déplacer un répertoire dans Visual Basic

La méthode My.Computer.FileSystem.MoveDirectory peut être utilisée pour déplacer des répertoires.

Si vous tentez de déplacer un répertoire vers un répertoire qui n'existe pas, le répertoire cible sera créé.

Si *overwrite* a la valeur **True** et que le répertoire de destination existe déjà, les nouveaux fichiers seront ajoutés aux fichiers existants dans le répertoire. Les fichiers du répertoire de destination seront remplacés si le répertoire source contient des fichiers portant le même nom.

Pour déplacer un répertoire

- Utilisez la méthode **MoveDirectory** en spécifiant les répertoires source et cible. L'exemple suivant déplace `Dir1` dans `Dir2`.

  VB

  ```
  My.Computer.FileSystem.MoveDirectory("C:\Dir1", "C:\Dir2")
  ```

Pour déplacer un répertoire et remplacer les répertoires existants

- Utilisez la méthode **MoveDirectory** en spécifiant les répertoires source et cible. L'exemple suivant déplace `Dir1` dans `Dir2` en ajoutant ses fichiers aux fichiers existants si le répertoire de destination existe déjà.

  VB

  ```
  My.Computer.FileSystem.MoveDirectory("C:\Dir1", "C:\Dir2", True)
  ```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès est **Nothing** (ArgumentNullException).
- La source n'est pas valide (DirectoryNotFoundException).
- La source est un répertoire racine (IOException).
- Le chemin d'accès combiné pointe sur un fichier existant (**IOException**).
- Le chemin source et le chemin cible sont identiques (**IOException**).
- Le fichier existe déjà et *overwrite* a la valeur **False** (**IOException**).
- Un sous-répertoire de fichier ne peut pas être copié (**IOException**).
- L'opération est cyclique (InvalidOperationException).
- Un nom de fichier ou de répertoire du chemin d'accès contient le signe deux-points (:) (NotSupportedException).
- *onUserCancel* a la valeur **UICancelOption.ThrowException**, et l'utilisateur annule l'opération (OperationCanceledException).
- *onUserCancel* a la valeur **UICancelOption.ThrowException**, et l'opération ne peut pas être effectuée (**OperationCanceledException**).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).

- *onUserCancel* a la valeur **UICancelOption.ThrowException** et l'utilisateur n'a pas les autorisations nécessaires (SecurityException).

- L'utilisateur n'est pas autorisé à créer un fichier (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : déplacer le contenu d'un répertoire dans Visual Basic
Comment : copier un répertoire dans un autre répertoire dans Visual Basic
Comment : renommer un répertoire dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Comment : déplacer le contenu d'un répertoire dans Visual Basic

Utilisez la méthode My.Computer.FileSystem.GetFiles pour obtenir la liste des fichiers d'un dossier et utilisez la méthode My.Computer.FileSystem.MoveFile pour déplacer des fichiers entre répertoires.

Si la structure cible n'existe pas lorsque **MoveFile** est appelé, elle sera créée.

Pour déplacer le contenu d'un répertoire

- Utilisez la méthode **GetFiles** pour obtenir la liste des fichiers du répertoire, suivie de la méthode **MoveFile** en fournissant le fichier source et le répertoire dans lequel vous le déplacez. Cet exemple déplace tous les fichiers du répertoire **Mes documents** vers le répertoire nommé `StorageDir`.

  VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.*")

    My.Computer.FileSystem.MoveFile(foundFile, "C:\StorageDir")
Next
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (commence par `\\.\`) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe sur un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : renommer un fichier dans Visual Basic
Comment : déplacer un fichier dans Visual Basic
Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic
Comment : déplacer un répertoire dans Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Comment : déplacer une collection de fichiers dans Visual Basic

Référence

My.Computer.FileSystem.MoveFile, méthode
My.Computer.FileSystem.GetFiles, méthode

Comment : récupérer le contenu du répertoire Mes Documents dans Visual Basic

L'objet My.Computer.FileSystem.SpecialDirectories peut être utilisé pour lire de nombreux répertoires **Tous les utilisateurs**, tels que **Mes documents** ou **Bureau**.

Pour lire le dossier Mes documents

- Utilisez la méthode **ReadAllText** pour lire le texte de chaque fichier dans un répertoire spécifique. Le code suivant spécifie un répertoire et un fichier, puis utilise **ReadAllText** pour les lire dans la chaîne nommée `patients`.

  VB

```
Dim path As String
Dim patients As String
path = My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\" & "Patients.txt"
patients = My.Computer.FileSystem.ReadAllText(path)
```

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet
Membres de l'objet My.Computer.FileSystem.SpecialDirectories
My.Computer.FileSystem.ReadAllText, méthode

Comment : analyser des chemins d'accès dans Visual Basic

L'objet My.Computer.FileSystem offre plusieurs méthodes utiles lors de l'analyse de chemins d'accès.

- La méthode My.Computer.FileSystem.CombinePath prend deux chemins d'accès et retourne un chemin d'accès combiné correctement mis en forme.
- The La méthode My.Computer.FileSystem.GetParentPath retourne le chemin d'accès absolu du parent du chemin d'accès fourni.
- La méthode My.Computer.FileSystem.GetFileInfo retourne un objet FileInfo qui peut être interrogé pour déterminer les propriétés du fichier, telles que son nom et son chemin d'accès.

Ne vous basez pas sur l'extension d'un nom de fichier pour en déterminer le contenu. Par exemple, il se peut qu'un fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Pour déterminer le nom et le chemin d'accès d'un fichier

- Utilisez les propriétés DirectoryName et Name de l'objet **FileInfo** pour déterminer le nom et le chemin d'accès d'un fichier. Cet exemple détermine le nom et le chemin d'accès, puis les affiche.

  VB

```
Dim testFile As System.IO.FileInfo
testFile = My.Computer.FileSystem.GetFileInfo("C:\TestFolder1\test1.txt")
Dim folderPath As String = testFile.DirectoryName
MsgBox(folderPath)
Dim fileName As String = testFile.Name
MsgBox(fileName)
```

Pour associer le nom et le répertoire d'un fichier afin de créer le chemin d'accès complet

- Utilisez la méthode **CombinePath** en fournissant le répertoire et le nom. Cet exemple prend les chaînes `folderPath` et `fileName` créées dans l'exemple précédent, les associe et affiche le résultat.

  VB

```
Dim fullPath As String
fullPath = My.Computer.FileSystem.CombinePath(folderPath, fileName)
MsgBox(fullPath)
```

Voir aussi

Tâches

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic
Comment : obtenir des informations à propos d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.CombinePath, méthode
My.Computer.FileSystem.GetFileInfo, méthode
FileInfo

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Cette section contient des rubriques relatives à l'obtention et à la définition de fichiers, de répertoires et de propriétés de lecteur.

Dans cette section

Comment : obtenir des informations à propos d'un fichier dans Visual Basic

Illustre comment obtenir des informations, telles que l'heure du dernier accès, sur un fichier.

Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic

Illustre comment déterminer le chemin d'accès absolu d'un fichier.

Comment : déterminer l'extension d'un fichier dans Visual Basic

Illustre comment déterminer l'extension d'un fichier.

Comment : déterminer l'heure de la dernière consultation d'un fichier dans Visual Basic

Illustre comment déterminer l'heure de la dernière consultation d'un fichier.

Comment : déterminer l'heure de dernière modification d'un fichier dans Visual Basic

Illustre comment déterminer l'heure de la dernière modification d'un fichier.

Comment : déterminer la taille d'un fichier dans Visual Basic

Illustre comment déterminer la taille d'un fichier en octets.

Comment : déterminer l'heure de création d'un fichier dans Visual Basic

Illustre comment déterminer la date de création d'un fichier.

Comment : déterminer si un fichier est masqué dans Visual Basic

Illustre comment déterminer si un fichier est masqué.

Comment : déterminer si un fichier est en lecture seule dans Visual Basic

Illustre comment déterminer si un fichier est en lecture seule.

Comment : déterminer les attributs d'un fichier dans Visual Basic

Illustre comment déterminer les attributs d'un fichier.

Comment : déterminer l'heure de création d'un répertoire dans Visual Basic

Illustre comment déterminer la date de création d'un répertoire.

Comment : déterminer si un répertoire est en lecture seule dans Visual Basic

Illustre comment déterminer si un répertoire est en lecture seule.

Comment : déterminer les attributs d'un répertoire dans Visual Basic

Illustre comment déterminer les attributs d'un répertoire.

Comment : déterminer si un fichier existe dans Visual Basic

Illustre comment déterminer si un fichier existe.

Comment : déterminer si un répertoire existe dans Visual Basic

Illustre comment déterminer si un répertoire existe.

Comment : déterminer le nom de volume d'un lecteur dans Visual Basic

Illustre comment déterminer le nom de volume d'un lecteur.

Comment : déterminer le format d'un lecteur dans Visual Basic

Illustre comment déterminer le type de système de fichiers d'un lecteur.

Comment : déterminer le type d'un lecteur dans Visual Basic

Illustre comment déterminer le type d'un lecteur.

Comment : déterminer l'espace total d'un lecteur dans Visual Basic

Illustre comment déterminer l'espace total d'un lecteur.

Comment : déterminer l'espace physique disponible sur un lecteur dans Visual Basic

Illustre comment déterminer l'espace physique disponible d'un lecteur.

Comment : déterminer le répertoire racine d'un lecteur dans Visual Basic

Illustre comment déterminer le répertoire racine d'un lecteur.

Comment : déterminer le répertoire système de Windows dans Visual Basic

Illustre comment déterminer l'emplacement du répertoire système de Windows.

Référence

My.Computer.FileSystem, objet

Décrit l'objet **My.Computer.FileSystem**, ses méthodes et ses propriétés.

My.Computer.FileSystem.DirectoryExists, méthode

Décrit la méthode **DirectoryExists**.

My.Computer.FileSystem.FileExists, méthode

Décrit la méthode **FileExists**.

My.Computer.FileSystem.GetDirectoryInfo, méthode

Décrit la méthode **GetDirectoryInfo**.

My.Computer.FileSystem.GetFileInfo, méthode

Décrit la méthode **GetFileInfo**.

My.Computer.FileSystem.GetParentPath, méthode

Décrit la méthode **GetParentPath**.

Rubriques connexes

FileInfo

Décrit l'objet **FileInfo**.

DirectoryInfo

Décrit l'objet **DirectoryInfo**.

FileAttributes

Décrit l'énumération **FileAttributes**.

Écriture dans des fichiers en Visual Basic

Répertorie les tâches relatives à l'écriture dans les fichiers.

Lecture à partir de fichiers en Visual Basic

Répertorie les tâches relatives la lecture de fichiers.

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Répertorie les tâches relatives à la création, à la suppression et au déplacement de répertoires et de fichiers.

Comment : obtenir des informations à propos d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour déterminer facilement les informations sur les propriétés d'un fichier. Les propriétés de l'objet **FileInfo** concernent les attributs, l'heure de création, le répertoire, le nom de répertoire (le cas échéant), l'extension, le nom complet, l'heure du dernier accès, l'heure de la dernière écriture, la longueur et le nom.

Une exception n'est pas levée si le fichier n'existe pas ; elle est plutôt levée lorsque vous accédez aux propriétés de l'objet pour la première fois.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concern e les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation d e paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Procédure

Pour obtenir des informations sur un fichier

1. Utilisez la méthode **GetFileInfo** pour récupérer un objet **FileInfo** qui peut être examiné pour déterminer ses propriétés. L'exemple suivant récupère un objet **FileInfo** pour le fichier `MyLogFile.log`.

   VB

   ```
   Dim information As System.IO.FileInfo
   information = My.Computer.FileSystem.GetFileInfo("C:\MyLogFile.log")
   ```

2. Examinez l'objet **FileInfo** pour extraire les informations dont vous avez besoin. Les lignes de code suivantes indiquent le nom complet, l'heure du dernier accès et la longueur du fichier.

   VB

   ```
   MsgBox("The file's full name is " & information.FullName & ".")
   MsgBox("Last access time is " & information.LastAccessTime & ".")
   MsgBox("The length is " & information.Length & ".")
   ```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le nom du chemin d'accès est incorrect. Il contient par exemple des caractères non valides ou se compose uniquement d'un espace blanc (ArgumentException).
- Le fichier n'existe pas ou est **Nothing** (ArgumentNullException).
- Le chemin d'accès contient le signe deux-points au milieu de la chaîne (NotSupportedException).
- Le chemin d'accès est trop long (PathTooLongException
- L'utilisateur n'a pas les autorisations nécessaires (SecurityException).
- L'utilisateur n'a pas l'accès ACL (liste de contrôle d'accès) au fichier (UnauthorizedAccessException).

Voir aussi

Tâches

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

FileInfo

Autres ressources

Accès au fichier avec Visual Basic

Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo retourne un objet FileInfo de la propriété FullName qui peut être utilisé pour déterminer les informations relatives à un fichier, y compris son emplacement.

Si un fichier n'existe pas, **GetFileInfo** ne lève pas d'exception, mais une exception est levée la première fois que vous accédez à une propriété de l'objet **System.IO.FileInfo**.

Procédure

Pour déterminer le chemin d'accès absolu d'un fichier

- Utilisez la méthode **GetFileInfo** afin de retourner un objet **FileInfo** pour le fichier que vous souhaitez examiner. La propriété **FullName** contient le chemin d'accès absolu. L'exemple suivant détermine le chemin d'accès absolu de `Test.txt` et l'affiche dans une boîte de message.

  VB

```
Dim getInfo As System.IO.FileInfo
getInfo = My.Computer.FileSystem.GetFileInfo("C:\TestFolder1\test.txt")
MsgBox(getInfo.FullName)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- le nom du chemin d'accès est incorrect. Il contient par exemple des caractères non valides ou se compose uniquement d'un espace blanc (ArgumentException).
- Le fichier n'existe pas ou est **Nothing** (ArgumentNullException).
- Le chemin d'accès contient le signe deux-points au milieu de la chaîne (NotSupportedException).
- Le chemin d'accès est trop long (PathTooLongException
- L'utilisateur n'a pas les autorisations nécessaires (SecurityException).
- L'utilisateur n'a pas l'accès ACL (liste de contrôle d'accès) au fichier (UnauthorizedAccessException).

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem.GetFileInfo, méthode

FileInfo

Comment : déterminer l'extension d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFiles retourne les noms des fichiers comme une collection en lecture seule de chaînes qui peuvent être analysées avec la fonction Split (Visual Basic).

Pour déterminer l'extension d'un fichier

- Récupérez le chemin d'accès ou le nom du fichier et la fonction **Split** pour déterminer l'extension en fournissant le séparateur. Cet exemple utilise la méthode **GetFiles** pour retourner la collection de noms de fichiers dans le répertoire `testDirectory` et indique l'extension de chaque fichier.

  VB

```
For Each foundFile As String In _
My.Computer.FileSystem.GetFiles("C:\TestDir")
    Dim check As String = _
    System.IO.Path.GetExtension(foundFile)
    MsgBox("The file extension is " & check)
Next
```

Voir aussi

Tâches

Comment : rechercher une chaîne dans un tableau de chaînes (Visual Basic)
Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem.GetFiles, méthode
Path

Comment : déterminer l'heure de la dernière consultation d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié.

Vous pouvez utiliser la propriété **LastAccessTime** ou **LastAccessTimeUtc** pour obtenir ou définir l'heure du dernier accès. **LastAccessTimeUtc** affiche l'heure au temps UTC (Coordinated Universal Time). Pour plus d'informations, consultez Chaînes de format de date et d'heure.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer l'heure de la dernière consultation d'un fichier

- Utilisez la méthode **GetFileInfo** pour retourner un objet **FileInfo** pour le fichier, qui peut être interrogé pour connaître des informations. Cet exemple obtient un objet **FileInfo** pour `Testfile.txt` et utilise la propriété **LastAccessTime** pour afficher l'heure du dernier accès.

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File was last accessed on " & infoReader.LastAccessTime)
```

Voir aussi

Tâches

Comment : déterminer les attributs d'un fichier dans Visual Basic

Comment : déterminer l'heure de création d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.GetFileInfo, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer l'heure de dernière modification d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié.

Vous pouvez utiliser la propriété **LastWriteTime** ou **LastWriteTimeUtc** pour obtenir ou définir l'heure de la dernière modification du fichier. **LastWriteTimeUtc** affiche l'heure au temps UTC (Coordinated Universal Time). Pour plus d'informations, consultez Chaînes de format de date et d'heure.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer l'heure de la dernière modification d'un fichier

- Utilisez la méthode **GetFileInfo** pour retourner un objet **FileInfo** pour le fichier, qui peut être interrogé pour connaître des informations. Cet exemple obtient un objet **FileInfo** pour `Testfile.txt` et utilise la propriété **LastWriteTime** pour afficher l'heure du dernier accès.

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File was last modified on " & infoReader.LastWriteTime)
```

Voir aussi

Tâches

Comment : déterminer les attributs d'un fichier dans Visual Basic

Comment : déterminer l'heure de création d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.GetFileInfo, méthode

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer la taille d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié.

Vous pouvez utiliser la propriété **Length** pour déterminer la taille du fichier en octets.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concern e les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation d e paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer la taille d'un fichier

- Utilisez la méthode **GetFileInfo** pour retourner un objet **FileInfo** pour le fichier, qui peut être interrogé pour connaître des informations. Cet exemple obtient un objet **FileInfo** pour `Testfile.txt` et utilise la propriété **Length** pour afficher la taille du fichier en octets.

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File is " & infoReader.Length & " bytes.")
```

Voir aussi

Tâches

Comment : déterminer les attributs d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.GetFileInfo, méthode

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer l'heure de création d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié.

Vous pouvez utiliser la propriété **CreationTime** ou **CreationTimeUtc** pour obtenir ou définir l'heure du dernier accès. **CreationTimeUtc** affiche l'heure au temps UTC (Coordinated Universal Time). Pour plus d'informations, consultez Chaînes de format de date et d'heure.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer l'heure de création d'un fichier

- Utilisez la méthode **GetFileInfo** pour retourner un objet **FileInfo** pour le fichier, qui peut être interrogé pour connaître des informations. Cet exemple obtient un objet **FileInfo** pour `Testfile.txt` et utilise la propriété **CreationTime** pour afficher l'heure de création.

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File was created on " & infoReader.CreationTime)
```

Voir aussi

Tâches

Comment : déterminer les attributs d'un fichier dans Visual Basic

Comment : déterminer l'heure de la dernière consultation d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.GetFileInfo, méthode

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer si un fichier est masqué dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié, notamment sur l'énumération FileAttributes.

Pour déterminer si un fichier est masqué

1. Obtenez un objet **FileInfo** pour le fichier que vous souhaitez examiner. Cet exemple obtient un objet **FileInfo** pour le fichier `Testfile.txt`.

   VB

   ```
   Dim infoReader As System.IO.FileInfo
   infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
   ```

2. Obtenez un objet **FileAttributes** de l'objet **FileInfo**. Cet exemple obtient **FileAttributes** de l'objet **FileInfo**.

   VB

   ```
   Dim attributeReader As System.IO.FileAttributes
   attributeReader = infoReader.Attributes
   ```

3. Interrogez **FileAttributes** pour déterminer si le fichier est masqué. Cet exemple détermine si le fichier est masqué et affiche un résultat en conséquence.

   VB

   ```
   If (attributeReader And System.IO.FileAttributes.Hidden) > 0 Then
       MsgBox("File is hidden!")
   Else
       MsgBox("File is not hidden!")
   End If
   ```

Voir aussi

Tâches

Comment : déterminer les attributs d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem, membres de l'objet

My.Computer.FileSystem.GetFileInfo, méthode

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer si un fichier est en lecture seule dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer si un fichier est en lecture seule

- Utilisez la méthode **GetFileInfo** pour retourner un objet **FileInfo** pour le fichier, qui peut être interrogé pour connaître des informations. Cet exemple obtient un objet **FileInfo** pour `Testfile.txt` et affiche le message approprié si la propriété **IsReadOnly** a la valeur **True**.

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
If infoReader.IsReadOnly = True Then
    MsgBox("File is readonly!")
End If
```

Voir aussi

Tâches

Comment : déterminer si un répertoire est en lecture seule dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.GetFileInfo, méthode

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer les attributs d'un fichier dans Visual Basic

La méthode My.Computer.FileSystem.GetFileInfo peut être utilisée pour obtenir un objetFileInfo qui contient des informations sur le fichier spécifié, notamment sur l'énumération FileAttributes.

Ce tableau affiche les membres de **FileAttributes**.

Membre	Description
Archive	L'état d'archivage du statut. Les applications utilisent cet attribut pour marquer les fichiers pour la sauvegarde ou la suppression.
Compressed	Le fichier est compressé.
Device	Ce membre n'est pas utilisé pour le moment.
Directory	Le fichier est un répertoire.
Encrypted	Toutes les données du fichier sont chiffrées.
Hidden	Le fichier est masqué et ne sera pas affiché dans une liste des répertoires ordinaires.
Normal	Le fichier n'a pas d'autres attributs définis.
NotContentIndexed	Le fichier ne sera pas indexé par le service d'indexation de contenu du système d'exploitation.
Offline	Le fichier est hors connexion. Les données du fichier ne sont pas immédiatement disponibles.
ReadOnly	Le fichier est en lecture seule.
ReparsePoint	Le fichier contient un point d'analyse qui est un bloc de données définies par l'utilisateur.
SparseFile	Le fichier est un fichier fragmenté. Les fichiers fragmentés sont généralement des fichiers volumineux qui contiennent des données constituées, pour la plupart, de zéros.
System	Le fichier est un fichier système. Le fichier fait partie du système d'exploitation ou est utilisé exclusivement par le système d'exploitation.
Temporary	Le fichier est temporaire. Les systèmes de fichiers essaient de conserver toutes les données dans la mémoire pour un accès plus rapide plutôt que de les envoyer vers le stockage de masse. Un fichier temporaire doit être supprimé par l'application dès qu'il n'est plus nécessaire.

Pour déterminer si un fichier est chiffré

1. Obtenez un objet **FileInfo** pour le fichier que vous souhaitez examiner. Cet exemple obtient un objet **FileInfo** pour le fichier `Testfile.txt`.

   VB

   ```
   Dim infoReader As System.IO.FileInfo
   infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
   ```

2. Obtenez un objet **FileAttributes** de l'objet **FileInfo**. Cet exemple obtient **FileAttributes** de l'objet **FileInfo**.

VB

```
Dim attributeReader As System.IO.FileAttributes
attributeReader = infoReader.Attributes
```

3. Query **FileAttributes**. Cet exemple détermine si le fichier est chiffré et affiche un résultat en conséquence.

VB

```
If (attributeReader And System.IO.FileAttributes.Encrypted) > 0 Then
    MsgBox("File is encrypted!")
Else
    MsgBox("File is not encrypted!")
End If
```

Voir aussi

Tâches

Comment : déterminer si un fichier est masqué dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.GetFileInfo, méthode

FileAttributes

FileInfo

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer l'heure de création d'un répertoire dans Visual Basic

La méthode My.Computer.FileSystem.GetDirectoryInfo retourne un objet DirectoryInfo qui peut être interrogé pour connaître les informations sur le répertoire.

Si le répertoire n'existe pas, une exception n'est pas levée tant que vous n'accédez pas à une propriété de l'objet **DirectoryInfo**.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concern e les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation d e paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer l'heure de création d'un répertoire

- Utilisez la méthode **GetDirectoryInfo** afin de récupérer un objet **DirectoryInfo** pour le répertoire et interrogez la propriété CreationTime. Cet exemple affiche **CreationTime** pour `C:\Documents and Settings`.

  VB

```
Dim getInfo As System.IO.DirectoryInfo
getInfo = My.Computer.FileSystem.GetDirectoryInfo _
("C:\Documents and Settings")
MsgBox("The directory was created at " & getInfo.CreationTime)
```

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Voir aussi

Tâches

Comment : déterminer si un répertoire existe dans Visual Basic
Dépannage : lecture et écriture dans des fichiers texte

Référence

My.Computer.FileSystem.GetDirectoryInfo, méthode
DirectoryInfo
CreationTime

Comment : déterminer si un répertoire est en lecture seule dans Visual Basic

La méthode My.Computer.FileSystem.GetDirectoryInfo retourne un objet DirectoryInfo dont la propriété Attributes peut être interrogée pour obtenir des informations sur le répertoire, notamment s'il est en lecture seule ou non.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer si un répertoire est en lecture seule

1. Utilisez la méthode **GetDirectoryInfo** pour retourner un objet **DirectoryInfo** pour le répertoire spécifié. Cet exemple retourne un objet **DirectoryInfo** pour le répertoire `TestDirectory`.

   VB

   ```
   Dim reader As System.IO.DirectoryInfo
   reader = My.Computer.FileSystem.GetDirectoryInfo("C:\testDirectory")
   ```

2. Interrogez la propriété **Attributes** de l'objet pour déterminer s'il est en lecture seule.

   VB

   ```
   If (reader.Attributes And System.IO.FileAttributes.ReadOnly) > 0 Then
       MsgBox("Directory is readonly!")
   End If
   ```

Exemple

L'exemple suivant qui présente l'extrait de code ci-dessus dans le formulaire complet détermine si le répertoire `testDirectory` est en lecture seule et indique le résultat dans un message.

VB

```
Dim reader As System.IO.DirectoryInfo
reader = My.Computer.FileSystem.GetDirectoryInfo("C:\testDirectory")
If (reader.Attributes And System.IO.FileAttributes.ReadOnly) > 0 Then
    MsgBox("File is readonly!")
End If
```

Compilation du code

Si le répertoire n'existe pas, une exception n'est pas levée tant que vous n'accédez pas à une propriété de l'objet **DirectoryInfo**.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).

- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Voir aussi

Tâches

Comment : déterminer les attributs d'un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.GetDirectoryInfo, méthode

Comment : déterminer les attributs d'un répertoire dans Visual Basic

La méthode My.Computer.FileSystem.GetDirectoryInfo retourne un objet DirectoryInfo dont la propriété Attributes peut être interrogée pour obtenir des informations sur le répertoire.

Le tableau suivant répertorie les membres de l'énumération FileAttributes utilisés par la propriété **Attributes**.

Membre	Valeur numérique	Description
ReadOnly	1	Le fichier est en lecture seule.
Hidden	2	Le fichier est masqué et par conséquent n'est pas inclus dans une liste de répertoires ordinaires.
System	4	Le fichier est un fichier système. Le fichier fait partie du système d'exploitation ou est utilisé exclusivement par le système d'exploitation.
Directory	16	Le fichier est un répertoire.
Archive	32	L'état d'archivage du statut. Les applications utilisent cet attribut pour marquer les fichiers pour la sauvegarde ou la suppression.
Device	64	Non utilisé.
Normal	128	Le fichier est normal et aucun autre attribut n'est défini. Cet attribut est valide uniquement s'il est utilisé seul.
Temporary	256	Le fichier est temporaire. Les systèmes de fichiers essaient de conserver toutes les données dans la mémoire pour un accès plus rapide. Un fichier temporaire doit être supprimé lorsqu'il devient inutile.
SparseFile	512	Le fichier est un fichier fragmenté. Les fichiers fragmentés sont généralement des fichiers volumineux qui contiennent des données constituées, pour la plupart, de zéros.
ReparsePoint	1024	Le fichier contient un point d'analyse qui est un bloc de données définies par l'utilisateur associé à un fichier ou à un répertoire.
Compressed	2048	Le fichier est compressé.
Offline	4096	Le fichier est hors connexion et les données ne sont pas immédiatement disponibles.
NotContentIndexed	8192	Le fichier ne sera pas indexé par le service d'indexation de contenu du système d'exploitation.
Encrypted	16384	Le fichier ou le répertoire est chiffré. Pour les fichiers, cela signifie que toutes les données du fichier sont chiffrées. Pour les répertoires, cela signifie que le chiffrement est la valeur par défaut pour les fichiers et les répertoires nouvellement créés.

Pour déterminer si un répertoire est masqué

- Utilisez la méthode **GetDirectoryInfo** pour retourner un objet **DirectoryInfo**. Cet exemple retourne **DirectoryInfo** pour le répertoire `TestDir`, obtient un objet **FileAttributes** de l'objet **DirectoryInfo** et le vérifie pour déterminer s'il est masqué ou non. Vous pouvez tester d'autres attributs de la même manière.

  VB

```
Dim checkFile As System.IO.DirectoryInfo
checkFile = My.Computer.FileSystem.GetDirectoryInfo("C:\TestDir")
Dim attributeReader As System.IO.FileAttributes
attributeReader = checkFile.Attributes

If (attributeReader And System.IO.FileAttributes.Hidden) > 0 Then
    MsgBox("Directory is hidden")
End If
```

Voir aussi

Tâches

Comment : déterminer les attributs d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem.GetDirectoryInfo, méthode

DirectoryInfo

FileAttributes

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Comment : déterminer si un fichier existe dans Visual Basic

Vous pouvez utiliser la méthode My.Computer.FileSystem.FileExists pour déterminer si un fichier spécifié existe ou non.

Si l'application ne dispose pas des autorisations nécessaires pour lire le fichier spécifié, la méthode **FileExists** retourne **False**, qu'il existe ou non un chemin d'accès, mais ne lève pas d'exception.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Procédure

Pour déterminer si un fichier existe

- Utilisez la méthode **My.Computer.FileSystem.FileExists**, en fournissant le chemin d'accès, pour déterminer si le fichier spécifié existe ou non. L'exemple suivant vérifie si oui ou non le fichier `Check.txt` existe et fournit les informations dans un message.

  VB

```
If My.Computer.FileSystem.FileExists("c://Check.txt") Then
    MsgBox("File found.")
Else
    MsgBox("File not found.")
End If
```

Voir aussi

Tâches

Comment : déterminer si un répertoire existe dans Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem.FileExists, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Comment : déterminer si un répertoire existe dans Visual Basic

Vous pouvez utiliser la méthode My.Computer.FileSystem.DirectoryExists pour déterminer si un répertoire spécifié existe ou non.

L'appel à **DirectoryExists** requiert FileIOPermission.

Cette méthode ne lève pas d'exceptions.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Procédure

Pour déterminer si un répertoire existe

- Utilisez la méthode **My.Computer.FileSystem.DirectoryExists**, en fournissant le chemin d'accès, pour déterminer si le répertoire spécifié existe ou non. L'exemple suivant détermine si le répertoire `C:\backup\logs` existe et vérifie ses propriétés.

  VB

```
Dim logDirectoryProperties As System.IO.DirectoryInfo
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    logDirectoryProperties = My.Computer.FileSystem.GetDirectoryInfo("C:\backup\logs")
End If
```

Voir aussi

Tâches

Comment : déterminer si un fichier existe dans Visual Basic
Dépannage : lecture et écriture dans des fichiers texte
Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem.DirectoryExists, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Comment : déterminer le nom de volume d'un lecteur dans Visual Basic

La méthode My.Computer.FileSystem.GetDriveInfo fournit un objet DriveInfo qui peut être interrogé pour obtenir des informations sur un lecteur. Vous pouvez utiliser la propriété VolumeLabel pour déterminer le volume du lecteur. Si vous essayez d'accéder à la propriété sans disposer d'autorisations suffisantes, une exception SecurityException est levée.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer le nom de volume d'un lecteur

- Utilisez la propriété **VolumeLabel** pour déterminer le volume du lecteur, comme illustré dans le code suivant.

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.VolumeLabel)
```

Voir aussi

Tâches

Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Comment : déterminer le format d'un lecteur dans Visual Basic
Comment : déterminer le type d'un lecteur dans Visual Basic

Référence

My.Computer.FileSystem.GetDriveInfo, méthode

Comment : déterminer le format d'un lecteur dans Visual Basic

La méthode My.Computer.FileSystem.GetDriveInfo fournit un objet DriveInfo qui peut être interrogé pour obtenir des informations sur un lecteur. Vous pouvez utiliser la propriété DriveFormat pour déterminer le format du lecteur. Si vous essayez d'accéder à la propriété sans disposer d'autorisations suffisantes, une exception SecurityException est levée.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer le format d'un lecteur

- Utilisez la propriété **DriveFormat** pour déterminer le format du lecteur, comme illustré dans le code suivant.

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.DriveFormat)
```

Voir aussi

Tâches

Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Comment : déterminer le nom de volume d'un lecteur dans Visual Basic
Comment : déterminer le type d'un lecteur dans Visual Basic

Référence

My.Computer.FileSystem.GetDriveInfo, méthode

Comment : déterminer le type d'un lecteur dans Visual Basic

La méthode My.Computer.FileSystem.GetDriveInfo fournit un objet DriveInfo qui peut être interrogé pour obtenir des informations sur un lecteur. Vous pouvez utiliser la propriété **DriveType** pour déterminer le type du lecteur. Si vous essayez d'accéder à la propriété sans disposer d'autorisations suffisantes, une exception SecurityException est levée.

Le tableau suivant présente les membres de l'énumération **DriveType**.

Membre	Valeur	Description
Unknown	0	Il est impossible de déterminer le type de lecteur.
CDRom	1	Le lecteur est un lecteur de CD-ROM. Aucune distinction n'est faite entre les lecteurs de CD-ROM en lecture seule et en lecture/écriture.
DVDRom	2	Le lecteur est un lecteur de DVD-ROM.
Fixed	3	Le lecteur a un média fixe (inamovible). Cette valeur inclut tous les disques durs, y compris ceux qui sont amovibles.
RamDisk	4	Le lecteur est un bloc de RAM (Random Access Memory) sur l'ordinateur local qui se comporte comme un lecteur de disque.
Remote	5	Lecteurs réseau. Il s'agit des lecteurs partagés sur un réseau.
Removable	6	Le lecteur a un support amovible. Il inclut toutes les disquettes.

Pour déterminer le type d'un lecteur

- Utilisez la propriété **DriveType** pour déterminer le type, comme illustré dans le code suivant.

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.DriveType.ToString)
```

Voir aussi

Tâches

Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Comment : déterminer le nom de volume d'un lecteur dans Visual Basic

Référence

My.Computer.FileSystem.GetDriveInfo, méthode
DriveInfo

Comment : déterminer l'espace total d'un lecteur dans Visual Basic

La méthode My.Computer.FileSystem.GetDriveInfo fournit un objet DriveInfo qui peut être interrogé pour obtenir des informations sur un lecteur. Vous pouvez utiliser la propriété **TotalSize** pour déterminer la quantité d'espace sur un lecteur. Si vous essayez d'accéder à la propriété sans disposer d'autorisations suffisantes, une exception SecurityException est levée.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer la quantité totale d'espace disque sur un lecteur

- Utilisez la propriété **TotalSize** pour déterminer la quantité d'espace totale sur le lecteur, comme illustré dans le code suivant :

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.TotalSize)
```

Voir aussi

Tâches

Comment : déterminer l'espace physique disponible sur un lecteur dans Visual Basic
Comment : déterminer le nom de volume d'un lecteur dans Visual Basic
Comment : déterminer le type d'un lecteur dans Visual Basic

Référence

My.Computer.FileSystem.GetDriveInfo, méthode
DriveInfo

Comment : déterminer l'espace physique disponible sur un lecteur dans Visual Basic

La méthode My.Computer.FileSystem.GetDriveInfo fournit un objet DriveInfo qui peut être interrogé pour obtenir des informations sur un lecteur. Vous pouvez utiliser la propriété **TotalFreeSpace** pour déterminer la quantité d'espace sur un lecteur. Si vous essayez d'accéder à la propriété sans disposer d'autorisations suffisantes, une exception SecurityException est levée.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer la quantité d'espace libre sur un lecteur

- Utilisez la propriété **TotalFreeSpace** pour déterminer le volume d'espace disponible, comme illustré dans le code suivant :

  VB

  ```
  Dim cdrive As System.IO.DriveInfo
  cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
  MsgBox("Total free space: " & CStr(cdrive.TotalFreeSpace))
  ```

  Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il est localisé dans **Système de fichiers - Traitement de lecteurs, de dossiers et de fichiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Voir aussi

Tâches

Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Comment : déterminer le format d'un lecteur dans Visual Basic
Comment : déterminer le type d'un lecteur dans Visual Basic

Référence

My.Computer.FileSystem.GetDriveInfo, méthode
DriveInfo

Comment : déterminer le répertoire racine d'un lecteur dans Visual Basic

La méthode My.Computer.FileSystem.GetDriveInfo fournit un objet DriveInfo qui peut être interrogé pour obtenir des informations sur un lecteur. Vous pouvez utiliser la propriété **RootDirectory** pour déterminer le répertoire racine. Si vous essayez d'accéder à la propriété sans disposer d'autorisations suffisantes, une exception SecurityException est levée.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déterminer le répertoire racine d'un lecteur

- Utilisez la propriété **RootDirectory** pour déterminer le répertoire racine, comme illustré dans le code suivant.

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.RootDirectory)
```

Voir aussi

Tâches

Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Comment : déterminer le type d'un lecteur dans Visual Basic

Référence

My.Computer.FileSystem.GetDriveInfo, méthode
DriveInfo

Comment : déterminer le répertoire système de Windows dans Visual Basic

Cet exemple définit une chaîne contenant le chemin d'accès au répertoire système de Windows.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concerne les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Exemple

VB

```
Dim systemDirectory As String
systemDirectory = System.Environment.SystemDirectory
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Système d'exploitation Windows > Informations système**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

La propriété retourne une valeur telle que "`c:\WINNT\System32`".

Sécurité

Ce code nécessite la classe FileIOPermission.

Voir aussi

Tâches

Comment : déterminer les attributs d'un répertoire dans Visual Basic

Référence

Environment.GetFolderPath Method

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Analyse des fichiers texte avec l'objet TextFieldParser

L'objet **TextFieldParser** vous permet d'analyser et de traiter de très gros fichiers structurés sous forme de colonnes de texte à largeur délimitée, tels que les fichiers journaux ou les informations sur les bases de données héritées (legacy). Analyser un fichier texte avec le **TextFieldParser** correspond à parcourir un fichier texte, alors que la méthode d'analyse pour extraire des champs de texte correspond aux méthodes de manipulation des chaînes utilisées pour réinitialiser les chaînes délimitées.

Analyse de différents types de fichiers texte

Les fichiers texte peuvent avoir des champs à largeur différente, délimités par un caractère tel qu'une virgule ou un espace de tabulation. Définissez **TextFieldType** et le séparateur, comme dans l'exemple suivant, qui utilise la méthode **SetDelimiters** pour définir un fichier texte délimité par des tabulations :

VB

```
testReader.SetDelimiters(vbTab)
```

D'autres fichiers texte peuvent avoir des largeurs de champ fixes. Dans de tels cas, vous devez affecter le **TextFieldType** à **FixedWidth** et définir les largeurs de chaque champ, comme dans l'exemple suivant. Cet exemple utilise la méthode **SetFieldWidths** pour définir les colonnes de texte : la première colonne a une largeur de 5 caractères, la deuxième de 10, la troisième de 11, tandis que la quatrième a une largeur variable.

VB

```
testReader.SetFieldWidths(5, 10, 11, -1)
```

Une fois le format défini, vous pouvez parcourir le fichier à l'aide de la méthode **ReadFields** pour traiter tour à tour chaque ligne.

Si un champ ne correspond pas au format spécifié, une exception MalformedLineException est levée. Le cas échéant, les propriétés **ErrorLine** et **ErrorLineNumber** contiennent le texte à l'origine de l'exception et le numéro de ligne de celui-ci.

Analyse de fichiers à plusieurs formats

La méthode **PeekChars** de l'objet **TextFieldParser** peut être utilisée pour vérifier chaque champ avant de le lire en vous permettant de définir plusieurs formats pour les champs et de réagir en conséquence. Pour plus d'informations, consultez Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic.

Voir aussi

Tâches

Dépannage des exceptions : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

Référence

My.Computer.FileSystem.OpenTextFieldParser, méthode
TextFieldParser, objet
TextFieldParser.PeekChars, méthode
TextFieldParser.ReadFields, méthode
TextFieldParser.CommentTokens, propriété
TextFieldParser.Delimiters, propriété
TextFieldParser.ErrorLine, propriété
TextFieldParser.ErrorLineNumber, propriété
TextFieldParser.FieldWidths, propriété
TextFieldParser.HasFieldsEnclosedInQuotes, propriété
TextFieldParser.LineNumber, propriété
TextFieldParser.TextFieldType, propriété
TextFieldParser.TrimWhiteSpace, propriété
TextFieldParser.SetDelimiters, méthode
TextFieldParser.SetFieldWidths, méthode

Codages de fichiers

Les codages de fichier, plus connus sous le nom codages de caractères, spécifient comment représenter les caractères lors du traitement de texte. Un codage peut être préférable à un autre compte tenu des caractères de langage qu'il gère ou non, bien qu'Unicode soit préféré habituellement.

Lors de la lecture ou de l'écriture de fichiers, les codages incorrects de fichiers correspondants peuvent causer des exceptions ou des résultats inexacts.

Types de codages

Unicode est le codage par défaut pour utiliser les fichiers. Unicode est une norme de codage de caractères mondial qui utilise des valeurs de code 16 bits pour représenter tous les caractères utilisés dans l'informatique moderne, y compris les symboles techniques et les caractères spéciaux utilisés dans la publication.

Les normes de codage de caractères antérieurs étaient composées de jeux de caractères traditionnels, tels que le jeu de caractères ANSI Windows qui utilise des valeurs de code de 8 bits, ou des combinaisons de valeurs de 8 bits, pour représenter les caractères utilisés dans un langage ou une zone géographique précis.

Classe de codage

La classe Encoding représente un codage de caractères. Ce tableau répertorie le type de codage disponible et décrit chacun d'entre eux.

Nom	Description
ASCIIEncoding	Représente un codage de caractère ASCII de caractères Unicode.
UnicodeEncoding	Représente un codage UTF-16 de caractères Unicode.
UTF32Encoding	Représente un codage UTF-32 de caractères Unicode.
UTF7Encoding	Représente un codage UTF-7 de caractères Unicode.
UTF8Encoding	Représente un codage UTF-8 de caractères Unicode.

Voir aussi

Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic

Dépannage : lecture et écriture dans des fichiers texte

Cette rubrique aborde les problèmes courants rencontrés lors de l'utilisation des fichiers texte et propose une approche pour chacun d'entre eux.

Problèmes courants

Les problèmes les plus courants rencontrés lors de l'utilisation des fichiers texte concernent les exceptions de sécurité, les codages de fichiers ou les chemins d'accès incorrects.

Exceptions de sécurité

Une exception SecurityException est levée lorsqu'une erreur de sécurité se produit. Cela est souvent est dû au fait que l'utilisateur n'a pas les autorisations nécessaires, ce qui peut être résolu par l'ajout d'autorisations ou l'utilisation des fichiers dans un stockage isolé. Pour plus d'informations, consultez Dépannage des exceptions : System.Security.SecurityException et Stockage isolé.

Codages de fichiers

Les codages de fichier, plus connus sous le nom codages de caractères, spécifient comment représenter les caractères lors du traitement de texte. Les caractères inattendus d'un fichier texte peuvent être dus à un codage incorrect. Pour la plupart des fichiers, un codage peut être préférable à un autre en fonction des caractères de langue qu'il gère, bien qu'Unicode soit généralement préféré. Pour plus d'informations, consultez Codages de fichiers et Encoding.

Chemins d'accès incorrects

Lorsque de l'analyse des chemins d'accès, notamment des chemins d'accès relatifs, il est facile de fournir des données incorrectes. Vous pouvez résoudre de nombreux problèmes en vérifiant que vous fournissez le chemin d'accès correct. Pour plus d'informations, consultez Comment : analyser des chemins d'accès dans Visual Basic.

Voir aussi

Référence

My.Computer.FileSystem, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

Procédure pas à pas : manipulation de fichiers à l'aide de méthodes du .NET Framework

Cette procédure pas à pas illustre comment ouvrir et lire un fichier à l'aide de la classe StreamReader, vérifier si une tentative d'accès à un fichier est en cours, rechercher une chaîne dans un fichier lu avec une instance de la classe **StreamReader** et écrire dans un fichier à l'aide de la classe StreamWriter.

Remarque
Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concern e les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation d e paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Création de l'application

Démarrez Visual Studio et commencez le projet par la création d'un formulaire que l'utilisateur pourra employer pour écrire dans le fichier désigné.

Pour créer le projet

1. Dans le menu **Fichier**, choisissez **Nouveau projet**.
2. Dans le volet **Nouveau projet**, cliquez sur **Application Windows**.
3. Dans la zone **Nom**, tapez **MyDiary**, puis cliquez sur **OK**.

   Visual Studio ajoute le projet à l'**Explorateur de solutions** et le **Concepteur Windows Forms** s'ouvre.
4. Ajoutez au formulaire les contrôles du tableau suivant et définissez les valeurs de propriété correspondantes.

Objet	Propriétés	Valeur
Button	**Nom** **Text**	**Submit** **Submit Entry**
Button	**Nom** **Texte**	**Effacer** **Clear Entry**
TextBox	**Nom** **Texte** **Multiligne**	**Entry** **Please enter something.** **False**

Écriture dans le fichier

Pour ajouter la possibilité d'écrire dans un fichier via l'application, utilisez la classe **StreamWriter**. **StreamWriter** est conçu pour la sortie de caractères dans un codage particulier, tandis que la classe Stream est conçue pour l'entrée et la sortie d'octets. Utilisez **StreamWriter** pour écrire des lignes d'informations dans un fichier texte standard. Pour plus d'informations sur la classe **StreamWriter**, consultez StreamWriter, classe.

Pour ajouter une fonctionnalité d'écriture

1. Dans le menu **Affichage**, choisissez **Code** pour ouvrir l'éditeur de code.
2. Étant donné que l'application fait référence à l'espace de noms System.IO, ajoutez les instructions ci-après tout au début de votre code, avant la déclaration de classe du formulaire (`Public Class Form1`).

   VB

```
Imports System
```

```
Imports System.IO
```

Avant d'écrire dans le fichier, vous devez créer une instance d'une classe **StreamWriter**.

3. Dans le menu **Affichage**, choisissez **Concepteur** pour retourner au **Concepteur Windows Forms**. Double-cliquez sur le bouton `Submit` pour créer un gestionnaire d'événements Click pour ce bouton, puis ajoutez le code suivant

   VB

```
Dim fw As StreamWriter
```

Remarque

L'environnement de développement intégré (IDE, Integrated Development Environment) de Visual Studio revient à l'éditeur de code et positionne le point d'insertion dans le gestionnaire d'événements, à l'emplacement où vous devez ajouter le code.

1. Pour écrire dans le fichier, utilisez la méthode Write de la classe **StreamWriter**. Ajoutez le code ci-dessous immédiatement après `Dim fw As StreamWriter`. La levée d'une exception en cas de fichier introuvable ne doit pas vous inquiéter, car le fichier est créé s'il n'existe pas.

   VB

```
Dim ReadString As String
Try
    'Pass the file path and name to the StreamWriter constructor.
    'Indicate that Append is True, so file will not be overwritten.
    fw = New StreamWriter("C:\MyDiary.txt", True)
    ReadString = Entry.Text
    fw.WriteLine(ReadString)
Finally
    'Close the file.
    fw.Close()
End Try
```

2. Vérifiez que l'utilisateur ne peut soumettre aucune entrée vide en ajoutant le code suivant immédiatement après `Dim ReadString As String`.

   VB

```
If (Entry.Text = "" Or Entry.Text = "Please enter something.") Then
    Entry.Text = "Please enter something."
    Return
End If
```

3. Étant donné qu'il s'agit d'un agenda, l'utilisateur veut assigner une date à chaque entrée. Insérez le code suivant après `fw = New StreamWriter("C:\MyDiary.txt", True)` pour affecter la variable `Today` à la date actuelle.

   VB

```
Dim Today As DateTime
Today = Now
fw.Write(CStr(Today))
fw.Write(ControlChars.CrLf)
```

4. Enfin, attachez du code pour effacer la **TextBox**. Ajoutez le code suivant à l'événement **Click** du bouton `Clear`.

VB

```
Entry.Text = ""
```

Ajout de fonctionnalités d'affichage à l'agenda

Dans cette section, vous ajoutez une fonctionnalité qui affiche la dernière entrée dans la **TextBox** `DisplayEntry`. Vous pouvez également ajouter un ComboBox qui présente les diverses entrées que l'utilisateur peut sélectionner pour les afficher dans la **TextBox** `DisplayEntry`. Une instance de la classe **StreamReader** lit `MyDiary.txt`. Comme la classe **StreamWriter**, **StreamReader** est conçu pour être utilisé avec les fichiers texte.

Pour cette section de la procédure pas à pas, ajoutez au formulaire les contrôles du tableau suivant et définissez les valeurs de propriété correspondantes.

Contrôle	Propriétés	Valeurs
TextBox	**Nom** **Visible** **Taille** **Multiligne**	**DisplayEntry** **False** **120,60** **True**
Button	**Nom** **Texte**	**Affichage** **Affichage**
Button	**Nom** **Texte**	**GetEntries** **Get Entries**
ComboBox	**Nom** **Texte** **Enabled**	**PickEntries** **Select an Entry** **False**

Pour remplir la zone de liste déroulante

1. La **ComboBox** `PickEntries` permet d'afficher les dates auxquelles un utilisateur soumet une entrée. Par conséquent, celui-ci peut sélectionner une entrée associée à une date déterminée. Créez un gestionnaire d'événements **Click** pour le bouton `GetEntries` et ajoutez le code suivant.

   VB

```
Dim fr As StreamReader
Dim FileString As String
FileString = ""
Try
    fr = New System.IO.StreamReader("C:\MyDiary.txt")
    PickEntries.Items.Clear()
    PickEntries.Enabled = True
    Do
        FileString = fr.ReadLine
        If IsDate(FileString) Then
            PickEntries.Items.Add(FileString)
        End If
    Loop Until (FileString Is Nothing)
Finally
    If fr IsNot Nothing Then
        fr.Close()
    End If
```

```
End Try
PickEntries.Enabled = True
```

2. Pour tester votre code, appuyez sur F5 pour compiler l'application et cliquez sur **Get Entries**. Cliquez sur la flèche de déroulement dans la **ComboBox** pour afficher les dates des entrées.

Pour sélectionner et afficher des entrées individuelles

1. Créez un gestionnaire d'événements **Click** pour le bouton `Display` et ajoutez le code suivant.

   VB

```
Dim fr As StreamReader
Dim ReadString As String
'Make sure ReadString begins empty.
ReadString = ""
Dim FileString As String
fr = New StreamReader("C:\MyDiary.txt")
'If no entry has been selected, show the whole file.
If PickEntries.Enabled = False Or PickEntries.SelectedText Is Nothing Then
    Do
        'Read a line from the file into FileString.
        FileString = fr.ReadLine
        'add it to ReadString
        ReadString = ReadString & ControlChars.CrLf & FileString
    Loop Until (FileString = Nothing)
Else
    'An entry has been selected, find the line that matches.
    Do

        FileString = fr.ReadLine
    Loop Until FileString = CStr(PickEntries.SelectedItem)
    FileString = CStr(PickEntries.SelectedItem) & ControlChars.CrLf
    ReadString = FileString & fr.ReadLine

    'Read from the file until EOF or another Date is found.
    Do Until ((fr.Peek < 0) Or (IsDate(fr.ReadLine)))
        ReadString = ReadString & fr.ReadLine
    Loop
End If
fr.Close()
DisplayEntry.Visible = True
DisplayEntry.Text = ReadString
```

2. Pour tester votre code, appuyez sur F5 pour compiler l'application et soumettez une entrée. Cliquez sur **Get Entries**, sélectionnez une entrée dans la **ComboBox**, puis cliquez sur **Display**. Le contenu de l'entrée sélectionnée s'affiche dans la **TextBox** `DisplayEntry`.

Autorisation aux utilisateurs de supprimer ou de modifier des entrées

Enfin, vous pouvez inclure des fonctionnalités supplémentaires qui permettent aux utilisateurs de supprimer ou de modifier une entrée à l'aide des boutons `DeleteEntry` et `EditEntry`. Ces deux boutons restent désactivés si aucune entrée n'est affichée.

Ajoutez au formulaire les contrôles du tableau suivant et définissez les valeurs de propriété correspondantes.

Contrôle	Propriétés	Valeurs

Button	**Nom** **Texte** **Enabled**	**DeleteEntry** **Delete Entry** **False**
Button	**Nom** **Texte** **Enabled**	**EditEntry** **Edit Entry** **False**
Button	**Nom** **Texte** **Enabled**	**SubmitEdit** **Submit Edit** **False**

Pour activer la suppression et la modification d'entrées

1. Ajoutez le code suivant à l'événement **Click** du bouton `Display` après `DisplayEntry.Text = ReadString`.

   VB

   ```
   DeleteEntry.enabled = True
   ```

2. Créez un gestionnaire d'événements **Click** pour le bouton `DeleteEntry` et ajoutez le code suivant.

   VB

   ```
   Dim fr As StreamReader
   Dim ReadString As String
   Dim WriteString As String
   Dim ConfirmDelete As MsgBoxResult
   fr = New StreamReader("C:\MyDiary.txt")
   ReadString = fr.ReadLine
   ' Read through the textfile
   Do Until (fr.Peek < 0)
       ReadString = ReadString & vbCrLf & fr.ReadLine
   Loop
   WriteString = Replace(ReadString, DisplayEntry.Text, "")
   fr.Close()
   ' Check to make sure the user wishes to delete the entry
   ConfirmDelete = MsgBox("Do you really wish to delete this entry?", _
     MsgBoxStyle.OKCancel)
   If ConfirmDelete = MsgBoxResult.OK Then
       File.Delete("C:\MyDiary.txt")
       Dim fw As StreamWriter = File.CreateText("C:\MyDiary.txt")
       fw.WriteLine(WriteString)
       fw.Close()
       ' Reset controls on the form
       DisplayEntry.Text = ""
       PickEntries.Text = ""
       PickEntries.Items.Clear()
       PickEntries.Enabled = False
       DeleteEntry.Enabled = False
   End If
   ```

3. Lorsqu'un utilisateur affiche une entrée, le bouton `EditEntry` devient disponible. Ajoutez le code suivant à l'événement

Click du bouton `Display` après `DisplayEntry.Text = ReadString`.

VB

```
EditEntry.Enabled = True
```

4. Créez un gestionnaire d'événements **Click** pour le bouton `EditEntry` et ajoutez le code suivant.

   VB

   ```
   Entry.Text = DisplayEntry.Text
   SubmitEdit.Enabled = True
   ```

5. Créez un gestionnaire d'événements **Click** pour le bouton `SubmitEdit` et ajoutez le code suivant

   VB

   ```
   Dim fr As StreamReader
   Dim ReadString As String
   Dim WriteString As String
   If Entry.Text = "" Then
       MsgBox("Use Delete to Delete an Entry")
       Return
   End If
   fr = New StreamReader("C:\MyDiary.txt")
   ReadString = fr.ReadLine
   Do Until (fr.Peek < 0)
       ReadString = ReadString & vbCrLf & fr.ReadLine
   Loop
   WriteString = Replace(ReadString, DisplayEntry.Text, Entry.Text)
   fr.Close()
   File.Delete("C:\MyDiary.txt")
   Dim fw As StreamWriter = File.CreateText("C:\MyDiary.txt")
   fw.WriteLine(WriteString)
   fw.Close()
   DisplayEntry.Text = Entry.Text
   Entry.Text = ""
   EditEntry.Enabled = False
   SubmitEdit.Enabled = False
   ```

Pour tester votre code, appuyez sur F5 pour compiler l'application. Cliquez sur **Get Entries**, sélectionnez une entrée, puis cliquez sur **Display**. L'entrée s'affiche dans la **TextBox** `DisplayEntry`. Cliquez sur **Edit Entry**. L'entrée s'affiche dans la **TextBox** `Entry`. Modifiez l'entrée dans **TextBox** `Entry`, puis cliquez sur **Submit Edit**. Ouvrez le fichier `MyDiary.txt` pour confirmer vos corrections. À présent, sélectionnez une entrée et cliquez sur **Delete Entry**. Lorsque le message MessageBox vous demande de confirmer l'opération, cliquez sur **OK**. Fermez l'application et ouvrez `MyDiary.txt` pour confirmer la suppression.

Voir aussi

Référence

StreamReader Class

StreamWriter Class

Autres ressources

Procédures pas à pas relatives au langage Visual Basic

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Cette procédure pas à pas présente les notions de base de l'E/S de fichier dans Microsoft Visual Basic 2005. Pour illustrer cette fonctionnalité, elle crée une petite application, FileExplorer, qui examine des fichiers texte dans un répertoire et fournit des informations telles que les attributs, la date et l'heure de dernier accès et les 80 premiers caractères du fichier. Elle comprend une option qui écrit des informations dans un fichier journal.

Remarque
Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide concern e les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation d e paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Création de l'application

Pour débuter le projet, créez un formulaire à partir duquel les utilisateurs pourront sélectionner un répertoire, un fichier dans ce répertoire et les informations à récupérer sur ce fichier.

Pour créer le projet

1. Dans le menu **Fichier**, cliquez sur **Nouveau Projet**.

   La boîte de dialogue **Nouveau projet** s'affiche.

2. Dans le volet **Types de projets**, cliquez sur **Projets Visual Basic**, et dans le volet **Modèles**, cliquez sur **Application Windows**.

3. Dans la zone **Nom**, tapez `FileExplorer` pour définir le nom de projet.

   Visual Studio ajoute le projet à l'**Explorateur de solutions** et le Concepteur Windows Forms s'ouvre.

4. Ajoutez au formulaire les contrôles répertoriés dans le tableau suivant et définissez les valeurs de propriété correspondantes.

Objet	Propriétés	Valeur
TextBox	**Nom** **Texte**	**txtDirectory** **Répertoire**
Button	**Nom** **Texte**	**btnSubmit** **Submit**
Button	**Nom** **Texte**	**btnExamine** **Examine**
ComboBox	**Nom** **Texte**	**lstFilePick** **Select A File**
CheckBox	**Nom** **Texte** **Activé**	**chkFileLength** **File Length** **True**

CheckBox	**Nom** **Texte** **Activé**	**chkLastAccess** **Last Access Time** **True**
CheckBox	**Nom** **Texte** **Activé**	**chkSave** **Save Results** **False**

Affichage du répertoire actif

L'application FileExplorer a besoin d'un point de départ. Par conséquent, la **TextBox** `txtDirectory` utilise la fonction **My.Computer.FileSystem.CurrentDirectory** pour retourner et afficher une chaîne représentant le chemin d'accès en cours.

Pour renvoyer le répertoire actif

1. Créez un gestionnaire d'événements pour `Form1_Load` en double-cliquant sur le formulaire.

   L'éditeur de code s'ouvre.

2. Ajoutez le code ci-après pour que la zone de texte **TextBox** `txtDirectory` affiche l'emplacement en cours.

   VB

   ```
   txtDirectory.Text = My.Computer.FileSystem.CurrentDirectory
   ```

3. Exécutez le programme pour vérifier que le chemin d'accès correct est retourné.

   La **TextBox** `txtDirectory` affiche le répertoire actif.

Changement de répertoire

Étant donné que l'utilisateur souhaitera peut-être sélectionner des fichiers dans un répertoire différent, l'application utilise la même propriété pour passer d'un répertoire à un autre. Pour changer de répertoire, l'utilisateur entre un nouveau chemin d'accès dans la **TextBox** `txtDirectory`.

Pour changer de répertoire

1. Créez un gestionnaire d'événements de type clic pour `btnSubmit` en double-cliquant sur le contrôle situé sur le formulaire.

   L'éditeur de code s'ouvre.

2. Ajoutez le code ci-après au gestionnaire d'événements de type clic.

   VB

   ```
   Dim NewPath As String
   ' NewPath holds the path the user has entered.
   NewPath = txtDirectory.Text
   ' Change the location to NewPath.
   My.Computer.FileSystem.CurrentDirectory = NewPath
   ```

Vérification de l'entrée d'un chemin d'accès valide

Utilisez une instruction **Try...Catch** pour intercepter les exceptions causées par la soumission d'un chemin d'accès vide ou incorrecte.

Pour garantir des chemins d'accès valides

1. Dans l'événement `btnSubmit_Click`, après la ligne de code `Dim NewPath As String`, ajoutez `Dim ErrorMessage As`

`String` sur une nouvelle ligne.

2. Avant la ligne de code `My.Computer.FileSystem.CurrentDirectory = NewPath`, ajoutez une instruction **Try** sur une ligne distincte comme ci-après. Si vous appuyez sur Retour, l'éditeur de code insère automatiquement les instructions `Catch ex As Exception` et `End Try`. Retirez-les ; vous ajouterez vos propres instructions à l'étape suivante.

   VB

   ```
   Try

   ```

3. Après la ligne de code `My.Computer.FileSystem.CurrentDirectory = NewPath`, ajoutez le code suivant.

   VB

   ```
       ' This checks to make sure the path is not blank.
   Catch ex As Exception When NewPath = ""
       ErrorMessage = "You must enter a path."
       ' This catches errors caused by a path that is not valid.
   Catch
       ErrorMessage = "You must enter a valid path.  If trying " & _
       "to access a different drive, remember to include the drive " & _
       "letter."
   Finally
       ' Display the error message only if one exists.
       If ErrorMessage <> Nothing Then
           MsgBox(ErrorMessage)
       End If
   End Try
   ```

Affichage du contenu du répertoire dans une liste déroulante

Pour permettre à l'application d'afficher le contenu du répertoire actif, vous pouvez utiliser la méthode **My.Computer.FileSystem.GetFiles** qui retourne une collection de chaînes représentant les noms des fichiers du répertoire. Vous pouvez utiliser des caractères génériques à l'aide de **GetFiles** pour ne sélectionner que les fichiers d'un modèle particulier. Dans cet exemple, seuls les fichiers avec l'extension .txt sont retournés.

Pour afficher le contenu du répertoire

1. Au début de l'événement `btnSubmit_Click`, insérez le code suivant.

   ```
   Dim fileList As System.Collections.ObjectModel.ReadOnlyCollection(Of String).
   ```

2. Après la ligne `End Try`, insérez le code suivant.

   VB

   ```
   fileList = My.Computer.FileSystem.GetFiles( _
       My.Computer.FileSystem.CurrentDirectory, _
       FileIO.SearchOption.SearchTopLevelOnly, "*.txt")
   For Each foundFile As String In fileList
       lstFilePick.Items.Add(foundFile)
   Next
   ```

   Les informations collectées s'affichent dans la **ComboBox** `lstFilePick`, dans lequel vous pouvez prélever un fichier déterminé pour l'examiner.

Testez l'application en l'exécutant d'abord avec un répertoire qui ne contient pas de fichier .txt, puis avec un répertoire qui en contient plusieurs. Dans le premier cas, l'application affiche le message d'erreur approprié. Dans le second, l'application affiche la liste du **ComboBox** de tous les fichiers .txt du répertoire indiqué dans la **TextBox** `txtDirectory`.

Permettre à un utilisateur de sélectionner un fichier pour l'examiner

Bien que la **ComboBox** `lstFilePick` affiche tous les fichiers d'un répertoire, un utilisateur souhaitera probablement sélectionner et examiner un fichier déterminé.

Pour permettre la sélection d'un fichier spécifique

- Créez un gestionnaire d'événements de type clic pour `btnExamine_Click` et ajoutez le code ci-après pour confirmer la sélection d'un fichier.

  VB

```
Dim thisFile As System.IO.FileInfo
thisFile = My.Computer.FileSystem.GetFileInfo(CStr(lstFilePick.SelectedItem))
```

Permettre à un utilisateur de déterminer les informations à collecter

Les fichiers s'affichent désormais dans la **ComboBox** `lstFilePick`. Du code supplémentaire permet à l'utilisateur de préciser les informations à présenter. Par exemple, un utilisateur peut souhaiter connaître simplement la date du dernier accès au fichier. Un autre utilisateur peut également souhaiter connaître la taille de ce fichier. Un utilisateur sélectionne ou désactive des cases à cocher (`chkLastAccess`, `chkFileLength`) pour personnaliser les résultats.

Pour afficher des informations spécifiques

1. Déclarez les variables suivantes au début de l'événement `btnExamine_Click`, après `(lstFilePick.SelectedItem)`.

   VB

```
Dim stringlength As String
stringLength = "The file's length, in bytes, is: "
Dim stringLastAccess As String
stringLastAccess = "The file was last accessed on: "
Dim LastAccess As Date
Dim Length As Long
Dim FirstLine As String = ""
Dim FinalString As String = ""
Dim NewName As String
NewName = CType(lstFilePick.SelectedItem, String)

If NewName = Nothing Then
    MsgBox("You must select a file to examine.")
    Exit Sub
End If
```

   La méthode **My.Computer.FileSystem.GetFileInfo** retourne un objet FileInfo qui peut être interrogé pour obtenir des informations sur un fichier.

2. Ajoutez le code suivant à la fin de l'événement `btnExamine_Click`:

   VB

```
' Check last access time.
If chkLastAccess.Checked = True Then
    LastAccess = thisFile.LastAccessTime
End If
```

La propriété LastAccessTime détermine la date et l'heure du dernier accès au fichier. La valeur Date renvoyée indique la date et l'heure de création ou de dernière modification du fichier.

1. Ajoutez le code suivant à la fin de l'événement `btnExamine_Click`:

VB

```
' Check Length.
If chkFileLength.Checked = True Then
    Length = thisFile.Length
End If
```

La propriété Length, qui détermine la longueur du fichier, retourne une valeur **Long** indiquant la longueur du fichier en octets.

Affichage des résultats

Pour terminer la fonctionnalité de l'application, un message **MsgBox** présente les informations collectées.

Pour afficher les résultats

1. À la fin de l'instruction `If` qui détermine si la case `chkLastAccess` **CheckBox** a été cochée, ajoutez le code ci-dessous avant le dernier `End If`.

   VB

   ```
   ' Add to the messagebox.
   FinalString = FinalString & stringLastAccess & LastAccess & "." _
   & vbCrLf
   ```

2. À la fin de l'instruction `If` qui détermine si la case `chkFileLength` **CheckBox** a été cochée, ajoutez le code ci-dessous avant le dernier `End If`.

   VB

   ```
   ' Add to the messagebox.
   FinalString = FinalString & stringlength & CStr(Length) & "." _
   & vbCrLf
   ```

3. À la fin de l'instruction `If` qui détermine si la case `chkFirstLine` **CheckBox** a été cochée, ajoutez le code ci-dessous avant le dernier `End If`.

   VB

   ```
   ' Add to the messagebox.
   FinalString &= FirstLine & vbCrLf
   ```

Enregistrement des résultats

Un utilisateur peut souhaiter enregistrer les résultats de l'examen d'un fichier. Par conséquent, ajoutez du code qui vérifie la présence d'un fichier journal, en crée un si nécessaire, puis écrit les résultats dans celui-ci.

Pour créer un fichier journal

- Ajoutez le code ci-dessous à la fin de l'événement `btnExamine_Click`.

  VB

  ```
  ' Check to see if results should be saved.
  If chkSave.Checked = True And FinalString <> "" Then
      My.Computer.FileSystem.WriteAllText("log.txt", FinalString, True)
  End If
  ```

Pour tester votre application

1. Dans le répertoire de votre choix, créez un fichier texte nommé `test.txt` contenant la première ligne suivante :

   « This is the first line of the first file. The FileExplorer application examines only text files. »

2. Dans le même répertoire, créez un deuxième fichier texte nommé test2.txt contenant la première ligne suivante :

   « This is the first line of the second file. The FileExplorer application examines only text files. »

3. Démarrez l'application.

4. Tapez un chemin d'accès non valide et cliquez sur **Submit**.

   Le message suivant s'affiche : « You must enter a valid path. If trying to access a different drive, remember to include the drive letter. »

5. Tapez le chemin du répertoire qui contient test.txt et cliquez sur **Submit**.

   La **ComboBox** `lstFilePick` affiche les fichiers texte.

6. Sélectionnez test.txt dans la **ComboBox** `lstFilePick`. Assurez-vous que toutes les cases sont cochées et cliquez sur **Examine**.

   Le formulaire de résultats inclut la date du dernier accès et la longueur.

7. Sélectionnez test2.txt dans la **ComboBox** `lstFilePick`, désactivez toutes les cases à cocher, puis cliquez sur **Examine**.

   Le message d'erreur suivant s'affiche : « No file attribute checkboxes selected. »

8. Sélectionnez **Last Access** et **Save Results**, puis cliquez sur **Examine**.

   Le formulaire de résultats affiche uniquement l'heure du dernier accès.

9. Fermez FileExplorer.

   Étant donné que vous avez vérifié l'option **Save Results**, FileExplorer génère un fichier journal nommé `log.txt` dans le même répertoire que les fichiers texte.

Pour vérifier le fichier journal

- Dans le répertoire actif, ouvrez `log.txt` et vérifiez que FileExplorer a consigné les informations appropriées.

Voir aussi

Référence

My.Computer.FileSystem.CurrentDirectory, propriété

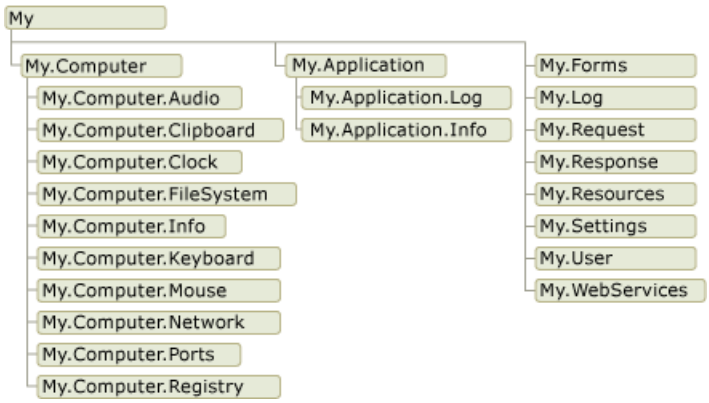
My.Computer.FileSystem.GetFileInfo, méthode

Développement avec My

Visual Basic fournit de nouvelles fonctionnalités destinées au développement rapide d'application. Celles-ci améliorent la productivité et facilitent l'exploitation, tout en fournissant de la puissance. L'une de ces fonctionnalités, **My**, donne accès à des informations et des instances d'objet par défaut mises en relation avec l'application et son environnement d'exécution. Ces informations sont organisées dans un format qui autorise une navigation IntelliSense et dont la logique s'organise en fonction de l'utilisation.

Les membres de niveau supérieur de **My** sont exposés comme des objets. Chaque objet se comporte comme un espace de noms ou une classe avec les membres **Shared**, et il expose un ensemble de membres associés.

Ce tableau affiche les objets **My** de niveau supérieur et les relations entre eux.



Dans cette section

Exécution de tâches avec My.Application, My.Computer et My.User

Décrit les trois objets **My** centraux, **My.Application**, **My.Computer**et **My.User** qui fournissent l'accès aux informations et aux fonctionnalités

Instances d'objets par défaut fournies par My.Forms et My.WebServices

Décrit les objets **My.Forms** et **My.WebServices** qui fournissent l'accès aux formulaires, aux sources de données et aux services Web XML utilisés par votre application.

Développement rapide d'application avec My.Resources et My.Settings

Décrit les objets **My.Resources** et **My.Settings** qui fournissent l'accès aux ressources et aux paramètres d'une application.

Vue d'ensemble du modèle d'application Visual Basic

Décrit le modèle de démarrage/arrêt d'application Visual Basic.

Comment My dépend du type de projet

Fournit des détails sur les fonctionnalités **My** disponibles dans les différents types de projet.

Voir aussi

Référence

My.Application, objet
My.Computer, objet
My.User, objet
My.Forms, objet
My.WebServices, objet

Concepts

Comment My dépend du type de projet

Exécution de tâches avec My.Application, My.Computer et My.User

Les trois objets **My** centraux qui fournissent l'accès aux informations et aux fonctionnalités couramment utilisées sont l'objet My.Application, l'objet My.Computer et l'objet My.User. Vous pouvez utiliser ces objets pour accéder aux informations liées à l'application actuelle, à l'ordinateur sur lequel est installée l'application, ou l'utilisateur actuel de l'application, respectivement.

My.Application, My.Computer et My.User

Les exemples suivants montrent comment les informations peuvent être récupérées à l'aide de **My**.

VB

```
' Displays a message box that shows the full command line for the
' application.
Dim args As String = ""
For Each arg As String In My.Application.CommandLineArgs
    args &= arg & " "
Next
MsgBox(args)
```

VB

```
' Gets a list of subfolders in a folder
My.Computer.FileSystem.GetDirectories _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments, True, "*Logs*")
```

Outre la récupération d'informations, les membres exposés à travers ces trois objets vous permettent également d'exécuter des méthodes en rapport avec cet objet. Par exemple, vous pouvez accéder à diverses méthodes destinées à la manipulation de fichiers ou à la mise à jour du Registre par le biais de **My.Computer**.

Les E/S de fichiers sont considérablement plus faciles et rapides avec **My**, qui inclut diverses méthodes et propriétés pour la manipulation de fichiers, répertoires et lecteurs. L'objet TextFieldParser vous permet de lire des fichiers volumineux structurés dotés de champs à largeur limitée ou fixe. Cet exemple ouvre **TextFieldParser** `reader` et l'utilise pour lire `C:\TestFolder1\test1.txt`.

VB

```
Dim reader As Microsoft.VisualBasic.FileIO.TextFieldParser
reader = My.Computer.FileSystem.OpenTextFieldParser _
("C:\TestFolder1\test1.txt")
reader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
reader.Delimiters = New String() {","}
Dim currentRow As String()
While Not reader.EndOfData
  Try
      currentRow = reader.ReadFields()
      Dim currentField As String
        For Each currentField In currentRow
            MsgBox(currentField)
        Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
          MsgBox("Line " & ex.Message & _
          "is not valid and will be skipped.")
    End Try
End While
```

My.Application vous permet de modifier la culture de votre application. L'exemple de code suivant montre comment appeler cette méthode.

VB

```
' Changes the current culture for the application to Jamaican English.
My.Application.ChangeCulture("en-JM")
```

Voir aussi

Référence

My.Application, objet
My.Computer, objet
My.User, objet

Concepts

Comment My dépend du type de projet

Instances d'objets par défaut fournies par My.Forms et My.WebServices

Les objets My.Forms et My.WebServices fournissent l'accès aux formulaires, sources de données et services Web XML utilisés par votre application. Pour ce faire, ils fournissent des collections d'*instances par défaut* de chacun de ces objets.

Instances par défaut

Une instance par défaut est une instance de la classe qui est fournie par le runtime ; elle n'a pas besoin d'être déclarée et instanciée à l'aide des instructions **Dim** et **New**. L'exemple suivant montre comment déclarer et instancier une instance d'une classe Form appelée `Form1`, et comment obtenir une instance par défaut de ce **Form** par le biais de **My.Forms**.

VB

```
' The old method of declaration and instantiation
Dim myForm As New Form1
myForm.show()
```

VB

```
' With My.Forms, you can directly call methods on the default
' instance()
My.Forms.Form1.Show()
```

L'objet **My.Forms** retourne une collection d'instances par défaut pour chaque classe **Form** qui existe dans votre projet. De la même façon, **My.WebServices** fournit une instance par défaut de la classe proxy pour chaque service Web auquel vous avez créé une référence dans votre application.

Voir aussi

Référence

My.Forms, objet

My.WebServices, objet

Concepts

Comment My dépend du type de projet

Développement rapide d'application avec My.Resources et My.Settings

L'objet **My.Resources** fournit l'accès aux ressources de l'application et vous permet de récupérer de manière dynamique des ressources pour votre application.

Récupération de ressources

Plusieurs ressources telles que les fichiers audio, les icônes, les images et les chaînes peuvent être récupérées par le biais de l'objet **My.Resources**. Par exemple, vous pouvez accéder aux fichiers de ressources spécifiques de la culture de l'application. L'exemple suivant définit l'icône du formulaire comme étant l'icône nommée `Form1Icon` stockée dans le fichier de ressources de l'application.

VB

```
Sub SetFormIcon()
    Me.Icon = My.Resources.Form1Icon
End Sub
```

L'objet **My.Resources** expose uniquement des ressources globales. Il ne fournit pas l'accès aux fichiers de ressources associés aux formulaires. Vous devez accéder aux ressources du formulaire à partir du formulaire. Pour plus d'informations, consultez Procédure pas à pas : localisation de Windows Forms.

De la même façon, l'objet **My.Settings** fournit l'accès aux paramètres de l'application et vous permet de stocker et de récupérer de manière dynamique des paramètres de propriété et d'autres informations pour votre application. Pour plus d'informations, consultez My.Resources, objet et My.Settings, objet.

Voir aussi

Référence

My.Resources, objet

My.Settings, objet

Concepts

Accès aux ressources de l'application

Accès aux paramètres d'application

Vue d'ensemble du modèle d'application Visual Basic

Visual Basic fournit un modèle précis pour contrôler le comportement des applications Windows Forms, à savoir le modèle d'application Visual Basic. Ce modèle inclut des événements pour gérer le démarrage et l'arrêt de l'application, ainsi que des événements pour l'interception des exceptions non gérées. Il fournit également une prise en charge pour le développement des applications à instance unique. Le modèle d'application est extensible, ce qui permet aux développeurs qui ont besoin de plus de contrôle de personnaliser ses méthodes substituables.

Utilisations du modèle d'application

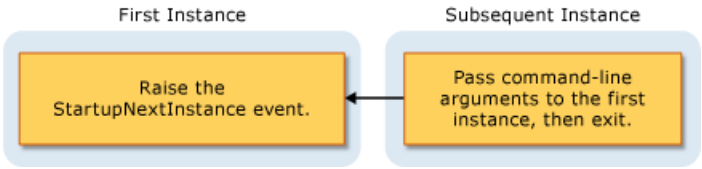
Une application classique doit effectuer des tâches lorsqu'elle démarre et s'arrête. Par exemple, lorsqu'elle démarre, l'application peut afficher un écran de démarrage, effectuer des connexions de base de données, charger un état enregistré, etc. Lorsque l'application s'arrête, elle peut fermer des connexions de base de données, enregistrer l'état actuel, etc. En outre, l'application peut exécuter du code spécifique lorsqu'elle s'arrête de manière inattendue, par exemple au cours d'une exception non gérée.

Le modèle d'application Visual Basic facilite la création d'une application à *instance unique*. Une application à instance unique est différente d'une application normale en ce sens qu'une seule instance de l'application peut être exécutée à la fois. Lorsque vous essayez de lancer une autre instance d'une application à instance unique, l'instance initiale reçoit une notification (par le biais de l'événement **StartupNextInstance**) qu'une autre tentative de lancement a été effectuée. La notification inclut les arguments de ligne de commande de l'instance suivante. L'instance suivante de l'application est ensuite fermée avant qu'une initialisation ne se produise.

Une application à instance unique démarre et vérifie s'il s'agit de la première instance ou d'une instance suivante de l'application :

- S'il s'agit de la première instance, elle démarre comme d'habitude.
- Toute tentative ultérieure de démarrer l'application, alors que la première instance est en cours d'exécution, produit un comportement très différent. La tentative suivante notifie la première instance à propos des arguments de ligne de commande, puis se ferme immédiatement. La première instance gère l'événement **StartupNextInstance** pour déterminer la nature des arguments de ligne de commande, puis continue son exécution.

  Ce diagramme montre comment une instance suivante signale la première instance.



En gérant l'événement **StartupNextInstance**, vous pouvez contrôler comment votre application à instance unique se comporte. Par exemple, Microsoft Outlook s'exécute en général comme une application à instance unique. Lorsque Outlook est en cours d'exécution et que vous essayez de le redémarrer, le focus passe à l'instance d'origine mais une autre instance ne s'ouvre pas.

Événements du modèle d'application

Les événements suivants figurent dans le modèle d'application :

- **Démarrage de l'application**. L'application déclenche l'événement Startup au démarrage. En gérant cet événement, vous pouvez ajouter du code qui initialise l'application avant que le formulaire principal soit chargé. L'événement **Startup** prévoit également l'annulation de l'exécution de l'application pendant cette phase du processus de démarrage.

  Vous pouvez configurer l'application pour qu'elle affiche un écran de démarrage pendant l'exécution du code de démarrage de l'application. Par défaut, le modèle d'application supprime l'écran de démarrage lorsque l'argument de ligne de commande `/nosplash` ou `-nosplash` est utilisé.

- **Applications à instance unique**. L'événement StartupNextInstance est déclenché lorsqu'une instance suivante de l'application à instance unique démarre. L'événement passe les arguments de ligne de commande de l'instance suivante.

- **Exceptions non gérées**. Si l'application rencontre une exception non gérée, elle déclenche l'événement UnhandledException. Le gestionnaire de cet événement peut examiner l'exception et déterminer s'il faut continuer l'exécution ou non.

L'événement **UnhandledException** n'est pas déclenché dans certains cas. Pour plus d'informations, consultez My.Application.UnhandledException, événement.

- **Changements de connectivité du réseau**. Si la disponibilité du réseau de l'ordinateur change, l'application déclenche l'événement NetworkAvailabilityChanged.

  L'événement **NetworkAvailabilityChanged** n'est pas déclenché dans certains cas. Pour plus d'informations, consultez My.Application.NetworkAvailabilityChanged, événement.

- **Arrêt de l'application**. L'application fournit l'événement Shutdown pour signaler à quel moment elle va s'arrêter. Dans ce gestionnaire d'événements, vous pouvez vérifier que les opérations que votre application doit exécuter (fermeture et enregistrement, par exemple) sont terminées. Vous pouvez configurer votre application pour qu'elle s'arrête en même temps que le formulaire principal ou uniquement lorsque tous les formulaires se ferment.

Disponibilité

Par défaut, le modèle d'application Visual Basic est disponible pour les projets Windows Forms. Si vous configurez l'application pour qu'elle utilise un objet de démarrage différent ou qu'elle démarre le code d'application avec un `Sub Main` personnalisé, cet objet ou classe devra éventuellement fournir une implémentation de la classe WindowsFormsApplicationBase pour utiliser le modèle d'application. Pour plus d'informations sur la modification de l'objet de démarrage, consultez Comment : modifier l'objet de démarrage pour une application.

Voir aussi

Référence

My.Application, objet
My.Application.Startup, événement
My.Application.StartupNextInstance, événement
My.Application.UnhandledException, événement
My.Application.Shutdown, événement
My.Application.NetworkAvailabilityChanged, événement
WindowsFormsApplicationBase

Concepts

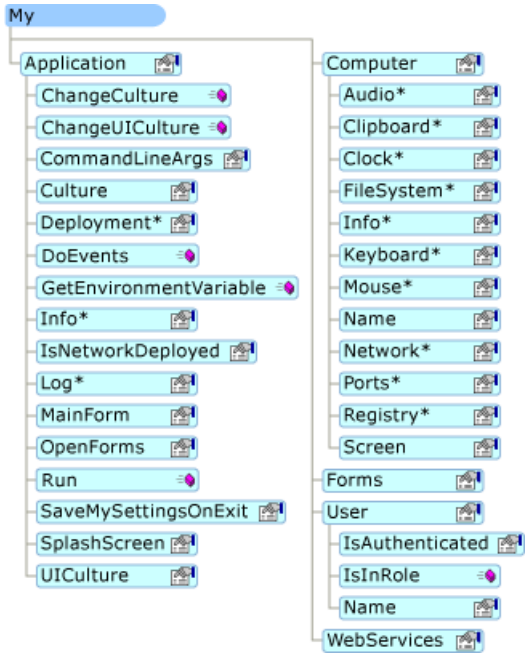
Extension du modèle d'application Visual Basic

Comment My dépend du type de projet

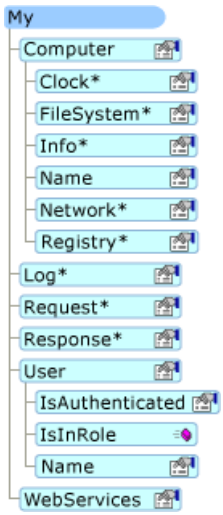
My expose uniquement les objets requis par un type de projet particulier. Par exemple, l'objet **My.Forms** est disponible dans une application Windows Forms et non dans une application console. Cette rubrique décrit les objets **My** qui sont disponibles dans différents types de projet.

My dans les applications Windows et les sites Web

My expose uniquement les objets qui sont utiles dans le type de projet actuel ; il supprime ceux qui sont non applicables. Par exemple, l'image suivante affiche le modèle d'objet **My** dans un projet Windows Forms.



Dans un projet de site Web, **My** expose les objets qui concernent un développeur Web (tel que les objets **My.Request** et **My.Response**) en supprimant ceux qui ne le sont pas (tels que l'objet **My.Forms**). L'image suivante affiche le modèle d'objet **My** dans un projet de site Web :



Détails du projet

Le tableau suivant affiche les objets **My** qui sont activés par défaut pour huit types de projet : application Windows, bibliothèque de classes, application console, bibliothèque de contrôles Windows, bibliothèque de contrôles Web, service Windows, vide et site Web.

Il existe trois versions de l'objet **My.Application**, deux versions de l'objet **My.Computer** et deux versions de l'objet **My.User**. Les détails de ces versions sont fournis dans les notes de bas de page après le tableau.

Objet My	Application Windows	Bibliothèque de classes	Application console	Bibliothèque de contrôles Windows	Bibliothèque de contrôles Web	Service Windows	Vide	Site Web

My.Application	**Oui** [1]	**Oui** [2]	**Oui** [3]	**Oui** [2]	Non	**Oui** [3]	Non	Non
My.Computer	**Oui** [4]	**Oui** [4]	**Oui** [4]	**Oui** [4]	**Oui** [5]	**Oui** [4]	Non	**Oui** [5]
My.Forms	**Oui**	Non	Non	**Oui**	Non	Non	Non	Non
My.Log	Non	Non	Non	Non	Non	Non	Non	**Oui**
My.Request	Non	Non	Non	Non	Non	Non	Non	**Oui**
My.Resources	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**	Non	Non
My.Response	Non	Non	Non	Non	Non	Non	Non	**Oui**
My.Settings	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**	Non	Non
My.User	**Oui** [6]	**Oui** [6]	**Oui** [6]	**Oui** [6]	**Oui** [7]	**Oui** [6]	Non	**Oui** [7]
My.WebServices	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**	Non	Non

[1] Version Windows Forms de **My.Application**. Dérive de la version de la console (voir Remarque 3) ; ajoute une prise en charge pour interagir avec les fenêtres de l'application et fournit le modèle d'application Visual Basic.

[2] Version de bibliothèque de **My.Application**. Fournit les fonctionnalités de base requises par une application : fournit des membres pour l'écriture dans le journal d'applications et l'accès aux informations de cette dernière.

[3] Version de console de **My.Application**. Dérive de la version de la bibliothèque (voir Remarque 2) et ajoute des membres supplémentaires pour accéder aux arguments de ligne de commande de l'application et aux informations de déploiement ClickOnce.

[4] Version Windows de **My.Computer**. Dérive de la version du serveur (voir Remarque 5) et fournit l'accès aux objets utiles d'un ordinateur client, tels que le clavier, l'écran et la souris.

[5] Version du serveur de **My.Computer**. Fournit des informations de base sur l'ordinateur, tels que le nom, l'accès à l'horloge, etc.

[6] Version Windows de **My.User**. Cet objet est associé à l'identité actuelle du thread.

[7] Version Web de **My.User**. Cet objet est associé à l'identité de l'utilisateur de la demande HTTP actuelle de l'application.

Voir aussi

Référence

/define (Visual Basic)
My.Application, objet
My.Computer, objet
My.Forms, objet
My.Log, objet
My.Request, objet
My.Response, objet
My.User, objet

My.WebServices, objet

Concepts

Personnalisation de la disponibilité ou non des objets dans My

Vue d'ensemble de la compilation conditionnelle

Accès aux données dans les applications Visual Basic

Visual Basic inclut plusieurs nouvelles fonctionnalités qui vous aident à développer des applications qui accèdent aux données. Vous pouvez créer des formulaires liés aux données pour des applications Windows en faisant glisser des éléments depuis la Sources de données, fenêtre jusqu'au formulaire. Vous liez des contrôles aux données en faisant glisser des éléments de la **fenêtre Sources de données** sur des contrôles existants.

Rubriques connexes

Création d'applications de données clientes

Fournit des liens vers des pages décrivant l'incorporation de fonctionnalités d'accès aux données dans vos applications.

Procédures pas à pas relatives aux données

Fournit des liens vers des pages décrivant des scénarios particuliers liés à l'accès aux données.

Mise en route de l'accès aux données

Fournit des liens vers les pages relatives à la création d'applications qui exploitent des données, à l'aide de Visual Studio.

Création d'applications de données avec Visual Studio

Traite des concepts fondamentaux que vous devez comprendre à propos de l'utilisation de données dans Visual Studio.

Connexion aux données dans Visual Studio

Fournit des liens vers des pages relatives à la connexion de votre application aux données avec des outils de conception et des objets de connexion ADO.NET, à l'aide de Visual Studio.

Préparation de votre application pour recevoir des données

Fournit des liens vers des pages qui expliquent ce que sont les groupes de données, comment en créer de nouveaux, ainsi que comment créer et modifier les objets dont ils sont constitués.

Extraction de données dans votre application

Fournit des liens vers des pages décrivant comment charger des données dans des groupes de données et comment exécuter les instructions SQL ainsi que des procédures stockées.

Affichage des données sur des formulaires dans les applications Windows

Fournit des liens vers des pages qui expliquent comment afficher des données dans des Windows Forms via des contrôles liés aux données.

Modification des données dans votre application

Fournit des liens vers les pages décrivant la manipulation des données dans les tables de données d'un groupe de données.

Validation des données

Fournit des liens vers des pages décrivant l'ajout d'une validation à un groupe de données lors de la modification de colonnes et de lignes.

Enregistrement des données

Fournit des liens vers des pages qui expliquent comment envoyer des données mises à jour d'une application vers la base de données.

Nouveautés des données

Fournit des informations et des liens vers de nouvelles fonctionnalités de données destinées aux applications clientes et de couches de données.

Nouveautés de Visual Basic

Aborde les nouvelles fonctionnalités de Visual Basic.

Nouveautés dans ADO.NET

Présente les nouvelles fonctionnalités d'ADO.NET.

Nouveautés de Visual Database Tools

Décrit les fonctionnalités nouvelles et améliorées de Visual Database Tools.

Nouveautés de l'accès aux données ASP.NET

Décrit les nouvelles fonctionnalités d'accès aux données dans les applications ASP.NET.

ADO.NET

Décrit les classes ADO.NET qui exposent des services d'accès aux données au programmeur .NET Framework.

Accès aux données ASP.NET (Visual Studio)

Fournit des liens vers des informations relatives à l'utilisation de données sur des pages Web ASP.NET.

Données dans les projets smart device managés

Décrit la manière de gérer les données des périphériques dans Visual Studio.

Données dans les solutions Office

Contient des liens vers des pages qui décrivent le fonctionnement des données dans les solutions Office, notamment des informations sur la programmation orientée schéma, la mise en cache de données et l'accès aux données côté serveur.

Débogage de l'application Visual Basic

Cette page fournit des pointeurs vers la documentation relative aux fonctions de débogage intégrées dans Visual Studio.

Vous pouvez utiliser la plus importante d'entre elles, à savoir le débogueur, pour observer le comportement de votre programme au moment de l'exécution et déterminer l'emplacement des erreurs sémantiques.

Avec le débogueur, vous pouvez examiner le contenu des variables de votre programme sans insérer des appels supplémentaires pour sortir les valeurs. De même, vous pouvez insérer un point d'arrêt dans votre code pour arrêter l'exécution à un point souhaité.

Contrôle de l'exécution

Le tableau suivant répertorie les tâches de débogage relatives au contrôle de l'exécution et pointe vers leurs pages d'aide associées.

Pour	Reportez-vous à
Interrompre l'exécution.	Comment : arrêter l'exécution
Démarrer l'exécution.	Comment : démarrer l'exécution
Effectuer le débogage au moment du design.	Procédure pas à pas : débogage au moment du design
Lancer le débogueur.	Comment : lancer automatiquement le débogueur
Effectuer un pas à pas détaillé uniquement dans le code que vous avez écrit et ignorer un autre code, tel que les appels au système.	Comment : effectuer un pas à pas détaillé dans Uniquement mon code
Effectuer un pas à pas détaillé dans le code.	Comment : effectuer un pas à pas détaillé dans le code
Arrêter le débogage.	Comment : arrêter le débogage ou l'exécution
Permettre le débogage juste-à-temps, qui lance le débogueur Visual Studio lorsqu'un programme qui s'exécute en dehors de Visual Studio rencontre une erreur irrécupérable.	Débogage juste-à-temps

Gestion des exceptions

Le tableau suivant répertorie les tâches de débogage relatives à la gestion des exceptions et pointe vers leurs pages d'aide associées.

Pour	Reportez-vous à
S'arrêter sur les exceptions non gérées.	Comment : s'arrêter sur les exceptions non gérées par l'utilisateur
S'arrêter lorsqu'une exception est levée.	Comment : arrêter l'exécution lorsqu'une exception est levée
S'arrêter sur les exceptions de première-chance.	Comment : arrêter l'exécution lorsqu'une exception est levée
Utiliser l'Assistant Exception.	Comment : corriger des erreurs d'exécution avec l'Assistant Exception
Ajouter une nouvelle exception.	Comment : ajouter de nouvelles exceptions
Poursuivre l'exécution après qu'une exception a été levée.	Poursuite de l'exécution à la suite d'une exception

Modifier & Continuer

Le tableau suivant répertorie les tâches de débogage relatives à la fonction Modifier & Continuer et pointe vers leurs pages d'aide associées.

Pour	Reportez-vous à
Activer et désactiver la fonction Modifier & Continuer.	Comment : activer et désactiver Modifier & Continuer
Arrêter l'application des modifications au code par la fonction Modifier & Continuer.	Comment : arrêter des modifications de code
Appliquer les modifications en mode arrêt.	Comment : appliquer des modifications en mode arrêt à l'aide de la fonctionnalité Modifier & Continuer

Examiner les données de débogage

Le tableau suivant répertorie les tâches de débogage relatives à l'affichage des données de débogage et pointe vers leurs pages d'aide associées.

Pour	Reportez-vous à
Utiliser la fenêtre **Registres** pour afficher le contenu du registre.	Comment : utiliser la fenêtre Registres
Utiliser la fenêtre **Pile des appels** pour afficher les appels de fonctions ou de procédures actuellement dans la pile.	Comment : utiliser la fenêtre Pile des appels
Utiliser la fenêtre **Code Machine** pour afficher le code assembleur correspondant aux instructions créées par le compilateur.	Comment : utiliser la fenêtre Code Machine
Utiliser la fenêtre **Modules** pour répertorier et décrire les modules utilisés par votre programme.	Comment : utiliser la fenêtre Modules
Utilisez la fenêtre **Explorateur de scripts** pour répertorier les fichiers de script actuellement chargés dans le programme.	Comment : utiliser la fenêtre Explorateur de scripts
Utiliser la fenêtre **Processus** pour afficher tous les processus attachés à Visual Studio ou lancés à partir de cette application.	Comment : utiliser la fenêtre Processus
Utiliser la fenêtre **Threads** pour examiner et contrôler les threads dans le programme.	Comment : utiliser la fenêtre Threads

Définition de points d'arrêt

Le tableau suivant répertorie les tâches de débogage relatives aux points d'arrêt et pointe vers leurs pages d'aide associées.

Pour	Reportez-vous à
Définir un point d'arrêt.	Comment : définir un point d'arrêt simple
Supprimer un point d'arrêt.	Comment : supprimer un point d'arrêt
Activer ou désactiver un point d'arrêt.	Comment : activer ou désactiver un point d'arrêt
Modifier l'emplacement d'un point d'arrêt.	Comment : modifier l'emplacement d'un point d'arrêt

Utiliser un filtre pour limiter le comportement des points d'arrêt à des ordinateurs, des processus et des threads spécifiés.	Comment : spécifier un filtre de point d'arrêt
Contrôler de manière conditionnelle si un point d'arrêt est exécuté ou non.	Comment : spécifier une condition de point d'arrêt
Utiliser la fenêtre **Points d'arrêt** pour accomplir des tâches relatives aux points d'arrêt.	Comment : utiliser la fenêtre Points d'arrêt

Voir aussi

Tâches

Procédure pas à pas : débogage d'un Windows Form

Comment : appliquer des modifications en mode arrêt à l'aide de la fonctionnalité Modifier & Continuer

Concepts

Débogage de SQL

Contrôle de l'exécution

Autres ressources

Débogage du code managé

Débogage du code natif

Débogage d'applications Web

Référence du débogage de l'interface utilisateur

Paramètres et préparation du débogage

Présentation du débogueur

Préparation du débogage : types de projets C#, J# et Visual Basic

Gestion des exceptions et des erreurs en Visual Basic

Visual Basic prend en charge la gestion structurée des exceptions (erreurs) qui permet au programme de détecter et parfois de récupérer les erreurs d'exécution. Visual Basic utilise une version améliorée de la syntaxe **Try...Catch...Finally** déjà prise en charge par d'autres langages tels que C++. La gestion structurée des exceptions associe une structure de contrôle moderne (semblable à **Select Case** ou **While**) à des exceptions, des blocs de code protégés et des filtres.

La gestion structurée des exceptions, qui est la méthode conseillée en matière de gestion d'erreurs en Visual Basic, facilite la création et la gestion de programmes comprenant des gestionnaires d'erreurs robustes et complets. Une gestion non structurée des exceptions à l'aide de **On Error** peut nuire aux performances de l'application et produire un code difficile à déboguer et à gérer.

Dans cette section

Introduction à la gestion des exceptions

Récapitule la gestion des exceptions dans vos programmes.

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Décrit les deux types de gestion des exceptions et explique laquelle est la plus appropriée selon la situation.

Types d'erreurs

Présente une vue d'ensemble des erreurs de syntaxe, des erreurs d'exécution et des erreurs logiques.

Correction automatique de la compilation intelligente

Décrit la fonctionnalité **Correction automatique de la compilation intelligente** et son utilisation.

Comment : corriger les erreurs de compilateur à l'aide de la correction automatique

Explique comment corriger les erreurs de compilation dans l'éditeur de code en utilisant la fonctionnalité **Correction automatique de la compilation intelligente**.

Configuration d'avertissements en Visual Basic

Explique comment activer et désactiver les avertissements du compilateur en Visual Basic.

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Décrit et illustre la gestion structurée des exceptions en Visual Basic.

Vue d'ensemble de la gestion non structurée des exceptions

Décrit et illustre la gestion non structurée des exceptions en Visual Basic.

Rubriques connexes

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Résume les modifications des éléments de langage en Visual Basic.

Présentation du débogueur

Décrit les notions fondamentales à connaître pour utiliser le débogueur Visual Studio. Ces rubriques comprennent les éléments fondamentaux du débogage, le contrôle de l'exécution, l'attachement à un programme en cours d'exécution, le débogage juste-à-temps, l'exécution automatique du débogueur, les dumps, les points d'arrêt, l'inspection de votre programme, la gestion des exceptions, Modifier & Continuer et l'utilisation d'expressions dans le débogueur.

Débogage juste-à-temps

Décrit le débogage juste-à-temps, une fonctionnalité qui lance automatiquement le débogueur Visual Studio lorsqu'un programme qui s'exécute en dehors de Visual Studio rencontre une erreur irrécupérable.

Débogage du code managé

Décrit les problèmes et les techniques de débogage courants pour les applications managées.

Assistant Exception

Décrit la fonctionnalité **Assistant Exception** qui facilite le dépannage des erreurs d'exécution.

Modifier & Continuer

Décrit la fonctionnalité **Modifier & Continuer**, qui vous permet d'apporter des modifications au code source lorsque le programme est en mode arrêt et de gagner ainsi du temps.

Débogage COM et ActiveX

Réunit des conseils sur le débogage des applications COM et des contrôles ActiveX.

Introduction à la gestion des exceptions

Visual Basic prend en charge la gestion *structurée* et *non structurée* des exceptions (erreurs). En plaçant du code de gestion des exceptions dans votre application, vous pouvez gérer la plupart des erreurs rencontrées par les utilisateurs et permettre ainsi à l'application de poursuivre son exécution. Vous pouvez utiliser la gestion des erreurs structurée et non structurée afin de réagir aux erreurs potentielles et de les empêcher d'interférer avec l'application.

Utilisez la gestion des exceptions dans les méthodes qui utilisent des opérateurs susceptibles de générer une exception, ou qui appellent ou accèdent à d'autres procédures susceptibles de générer une exception.

Si une exception se produit dans une méthode qui n'est pas conçue pour la gérer, l'exception est retournée à la méthode appelante ou à la méthode précédente. Si la méthode précédente n'a pas non plus de gestionnaire d'exceptions, l'exception est retournée à l'appelant de cette méthode, etc. La recherche d'un gestionnaire se poursuit dans la pile des appels, qui correspond à la série de procédures appelées dans l'application. Si aucun gestionnaire n'est trouvé pour l'exception, un message d'erreur s'affiche et l'application est interrompue.

Remarque
Une méthode peut contenir une gestion structurée ou non structurée des exceptions, mais pas les deux en même temps.

Gestion structurée des exceptions

Dans la gestion structurée des exceptions, les blocs de code sont encapsulés, chaque bloc étant associé à un ou plusieurs gestionnaires. Chaque gestionnaire spécifie une sorte de condition de filtrage sur le type d'exception qu'il gère. Lorsqu'une exception est déclenchée par du code dans un bloc protégé, les gestionnaires correspondants sont recherchés de manière ordonnée, et le premier gestionnaire à satisfaire la condition de filtrage est exécuté. Une seule méthode peut avoir plusieurs blocs de gestion structurée des exceptions, et les blocs peuvent également être imbriqués les uns dans les autres.

L'instruction **Try...Catch...Finally** sert de manière spécifique à la gestion structurée des exceptions. Pour plus d'informations, consultez Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Gestion non structurée des exceptions

L'instruction **On Error** sert de manière spécifique à la gestion non structurée des exceptions. Dans la gestion non structurée des exceptions, l'instruction **On Error** est placée au début d'un bloc de code. Elle a une « portée » qui s'étend à l'ensemble du bloc, car elle gère toutes les erreurs survenues dans ce bloc. Si le programme rencontre une autre instruction **On Error**, cette dernière devient valide et remplace la première instruction qui devient non valide. Pour plus d'informations, consultez Vue d'ensemble de la gestion non structurée des exceptions.

Voir aussi

Tâches

Résolution des problèmes liés à la gestion des exceptions

Procédure pas à pas : gestion structurée des exceptions

Référence

On Error, instruction (Visual Basic)

Concepts

Types d'erreurs

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Vue d'ensemble de la gestion non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

La gestion structurée des exceptions désigne simplement la création d'un mécanisme de gestion des exceptions à l'aide d'une structure de contrôle contenant des exceptions, de blocs de code isolés et de filtres. Cette fonctionnalité permet à votre code de différencier les différents types d'erreurs et de réagir en conséquence. Dans la gestion non structurée des exceptions, une instruction **On Error** placée au début du code gère toutes les exceptions.

Notes

La gestion structurée des exceptions est beaucoup plus polyvalente, robuste et souple que la gestion non structurée. Utilisez si possible la gestion structurée des exceptions. Toutefois, vous pouvez utiliser la gestion non structurée des exceptions dans les cas suivants :

- vous mettez à niveau une application écrite dans une version antérieure de Visual Basic ;
- vous développez une version préliminaire ou un projet de version d'une application et l'échec éventuel d'un arrêt progressif du programme ne vous poserait pas de problème particulier ;
- vous connaissez à l'avance la cause exacte de l'exception qui sera provoquée ;
- une date butoir approche, vous devez prendre des raccourcis et vous êtes disposé à sacrifier la souplesse pour la rapidité ;
- le code est simple ou si court que vous n'avez besoin de tester que la branche de code qui génère l'exception ;
- vous devez utiliser l'instruction **Resume Next**, qui n'est pas prise en charge dans la gestion structurée des exceptions.

Vous ne pouvez pas combiner la gestion structurée et la gestion non structurée des exceptions dans une même fonction. Si vous utilisez une instruction **On Error**, vous ne pouvez pas utiliser d'instruction **Try...Catch** dans la même fonction.

Quel que soit le mode de gestion des exceptions choisi, vous devez examiner au préalable les hypothèses associées au code. Par exemple, lorsque votre application demande à l'utilisateur d'entrer un numéro de téléphone, les hypothèses suivantes entrent en jeu :

- l'utilisateur entre un nombre, et non des caractères ;
- ce nombre présente un certain format ;
- l'utilisateur n'entre pas de chaîne vide (null) ;
- l'utilisateur possède un seul numéro de téléphone.

Les données entrées par utilisateur enfreignent peut-être l'une ou l'ensemble de ces hypothèses. Un code robuste exige une gestion adéquate des exceptions, pour que votre application puisse effectuer une reprise correcte après une infraction.

Sauf si vous pouvez garantir qu'une méthode ne lèvera jamais d'exception, prévoyez une gestion informative des exceptions. La gestion des exceptions doit être pertinente. Les messages ne doivent pas simplement signaler qu'une erreur s'est produite, mais ils doivent également identifier la cause et l'emplacement de l'erreur. Un message sans information pertinente ainsi que des lignes, telles que « Une erreur s'est produite » ne font que frustrer l'utilisateur.

Voir aussi

Tâches

Résolution des problèmes liés à la gestion des exceptions

Concepts

Types d'erreurs

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Vue d'ensemble de la gestion non structurée des exceptions

Types d'erreurs

En Visual Basic, les erreurs (également appelées *exceptions*) se rangent dans l'une des trois catégories suivantes : erreurs de syntaxe, erreurs d'exécution et erreurs logiques.

Erreurs de syntaxe

Les *erreurs de syntaxe* apparaissent au cours de l'écriture du code. Visual Basic vérifie votre code au fur et à mesure que vous le tapez dans la fenêtre de l'**éditeur de code** et vous signale vos erreurs comme la mauvaise orthographe d'un mot ou l'utilisation incorrecte d'un élément de langage. Les erreurs de syntaxe représentent le type d'erreur le plus fréquent. Vous pouvez les résoudre facilement dans l'environnement de codage dès qu'elles se produisent.

Remarque

L'instruction **Option Explicit** permet d'éviter des erreurs de syntaxe. Elle vous force à déclarer à l'avance toutes les variables qui doivent être utilisées dans l'application. Par conséquent, lorsque ces variables sont utilisées dans le code, les erreurs typo graphiques sont décelées immédiatement et peuvent être résolues.

Erreurs d'exécution

Les *erreurs d'exécution* sont celles qui apparaissent uniquement après la compilation et l'exécution de votre code. Elles sont liées à un code qui semble correct parce qu'il ne présente pas d'erreurs de syntaxe, mais dont l'exécution est impossible. Par exemple, vous pouvez écrire correctement une ligne de code permettant d'ouvrir un fichier. Mais si le fichier est endommagé, l'application ne peut pas exécuter la fonction **Open**, et son exécution est interrompue. Vous pouvez résoudre la plupart des erreurs d'exécution en réécrivant le code erroné, en le recompilant et en le ré-exécutant.

Erreurs logiques

Les *erreurs logiques* sont celles qui apparaissent lorsque l'application est en cours d'utilisation. Elles correspondent le plus souvent à des résultats non souhaités ou inattendus en réponse à des actions de l'utilisateur. Par exemple, le fait d'appuyer sur une mauvaise touche ou une influence extérieure peuvent entraîner l'arrêt de votre application dans les paramètres prévus ou de manière complète. Les erreurs logiques sont les plus difficiles à résoudre, car il n'est pas toujours facile de déterminer leur origine.

Voir aussi

Concepts

Introduction à la gestion des exceptions
Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic
Vue d'ensemble de la gestion non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions
Présentation du débogueur

Correction automatique de la compilation intelligente

La fonctionnalité **Correction automatique de la compilation intelligente** propose des corrections lorsque des erreurs se produisent et vous permet de choisir une solution à appliquer au code.

Lorsqu'une erreur se produit et qu'un symbole se présente sous le côté droit du tilde, positionnez la souris sur le tilde ou double-cliquez sur le message d'erreur dans la liste des tâches pour transformer le symbole en un panneau de balises actives. Lorsque vous cliquez sur le panneau des balises actives ou que vous positionnez le pointeur de la souris dessus, la fenêtre d'aide **Options de correction d'erreurs** s'ouvre et affiche une description de l'erreur ainsi qu'un ensemble de suggestions pour la résoudre. La fenêtre d'aide vous permet d'afficher un aperçu de la solution dans une fenêtre distincte, le cas échéant.

Une fois qu'elles sont appliquées, vous pouvez annuler les modifications à l'aide d'une opération d'annulation (CTRL+Z).

Pour fermer la fenêtre d'aide **Correction automatique de la compilation intelligente**, cliquez en dehors de l'éditeur.

Le tableau suivant répertorie et explique les éléments affichés dans la fenêtre **Correction automatique de la compilation intelligente** :

Élément	Description
Message d'erreur	Affiche le message associé à l'erreur.
Correction proposée	Affiche les solutions possibles pour corriger l'erreur. Cliquez sur une correction pour l'appliquer à v otre code.
Aperçu	Affiche un aperçu du code corrigé. Si la correction n'affecte pas le code, un aperçu ne s'affiche pas.
Développer tous les aper çus	L'activation de cette zone développe tous les aperçus.

Voir aussi

Tâches

Comment : corriger les erreurs de compilateur à l'aide de la correction automatique

Concepts

Types d'erreurs

Configuration d'avertissements en Visual Basic

Comment : corriger les erreurs de compilateur à l'aide de la correction automatique

La fonctionnalité Correction automatique de la compilation intelligente propose des corrections lorsque des erreurs se produisent et vous permet de choisir une solution à appliquer au code.

Pour corriger une erreur à l'aide de la fonctionnalité Correction automatique de la compilation intelligente

1. Lorsqu'une erreur pouvant être corrigée par la fonction Correction automatique de la compilation intelligente se produit, un glyphe s'affiche sous le côté droit du tilde. Lorsque vous positionnez le pointeur de la souris sur le tilde, le symbole est transformé en un panneau de balises actives.
2. Cliquez sur le panneau des balises actives ou positionnez le pointeur de la souris dessus. La fenêtre d'aide **Options de correction d'erreurs** ouvre et affiche une description de l'erreur ainsi qu'un ensemble de propositions pour la corriger. Cliquez sur la correction que vous souhaitez appliquer
3. Vous pouvez annuler les modifications qui ont été appliquées à l'aide d'une opération **Annuler** (CTRL+Z). Pour fermer la fenêtre d'aide **Options de correction d'erreurs**, cliquez en dehors de l'éditeur.

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs du compilateur en Visual Basic

Référence

Correction automatique de la compilation intelligente

Configuration d'avertissements en Visual Basic

Le compilateur Visual Basic inclut un ensemble d'avertissements sur le code susceptible de provoquer des erreurs d'exécution. Vous pouvez utiliser ces informations pour écrire un code plus clair, plus rapide, meilleur avec moins de bogues. Par exemple, le compilateur génère un avertissement lorsque l'utilisateur essaie d'appeler un membre d'une variable objet non assignée, quitte une fonction sans définir la valeur de retour ou exécute un bloc **Try** avec des erreurs dans la logique pour intercepter des exceptions.

Parfois, le compilateur fournit une logique supplémentaire au nom de l'utilisateur de sorte que ce dernier puisse se concentrer sur la tâche à exécuter, plutôt que sur l'anticipation d'erreurs possibles. Dans les versions précédentes de Visual Basic, **Option Strict** était utilisé pour limiter la logique supplémentaire fournie par le compilateur Visual Basic. La configuration des avertissements vous permet de limiter cette logique de manière plus granulaire au niveau des avertissements.

Vous pouvez personnaliser votre projet et désactiver certains avertissements non appropriés pour votre application tout en transformant les autres avertissements en erreurs. Cette page explique comment activer et désactiver les avertissements.

Activation et désactivation des avertissements

Il existe deux moyens de configurer les avertissements : vous pouvez les configurer à l'aide du **Concepteur de projets** ou utiliser les options du compilateur **/warnaserror** et **/nowarn**.

L'onglet **Compiler** de la page **Concepteur de projets** vous permet d'activer et de désactiver des avertissements. Sélectionnez la case à cocher **Désactiver tous les avertissements** pour désactiver tous les avertissements ; sélectionnez **Considérer tous les avertissements comme des erreurs** pour traiter tous les avertissements comme des erreurs. Certains avertissements peuvent être affichés ou masqués comme erreur ou avertissement dans le tableau affiché.

Lorsque **Option Strict** a le paramètre **Inactif**, les avertissements relatifs à **Option Strict** ne peuvent pas être traités indépendamment les uns des autres. Lorsque **Option Strict** a le paramètre **Actif**, les avertissements associés sont traités comme erreurs, quel que soit leur état. Lorsque **Option Strict** a le paramètre **Personnalisé** en spécifiant **/optionstrict:custom** dans le compilateur de ligne de commande, les avertissements **Option Strict** peuvent être affichés ou masqués de manière indépendante.

L'option de ligne de commande **/warnaserror** du compilateur permet également de spécifier si les avertissements sont traités comme des erreurs. Vous pouvez ajouter une liste séparée par des virgules à cette option pour spécifier quels avertissements doivent être traités comme des erreurs ou des avertissements à l'aide de + ou -. Le tableau suivant présente en détail les options possibles.

Option de ligne de commande	Informations fournies
`/warnaserror+`	Considérer tous les avertissements comme des erreurs
`/warnsaserror-`	Ne pas traiter les avertissements comme des erreurs. Il s'agit de l'option par défaut.
`/warnaserror+:<warning list >`	Traiter les avertissements spécifiques comme des erreurs, répertoriés par leur numéro d'ID d'erreur dans une liste séparée par des virgules.
`/warnaserror-:<warning list>`	Ne pas traiter les avertissements spécifiques comme des erreurs, répertoriés par leur numéro d'ID d'erreur dans une liste séparée par des virgules.
`/nowarn`	Ne pas signaler les avertissements.
`/nowarn:<warning list>`	Ne pas signaler les avertissements spécifiés, répertoriés par leur numéro d'ID d'erreur dans une liste séparée par des virgules.

La liste d'avertissements contient les numéros d'ID d'erreur des avertissements qui doivent être traités comme des erreurs et peuvent être utilisés avec les options de ligne de commande pour activer et désactiver des avertissements spécifiques. Si la liste d'avertissements contient un numéro incorrect, une erreur est signalée.

Exemples

Ce tableau d'exemples d'arguments de ligne de commande décrit le rôle de chaque argument.

Argument	Description
`vbc /warnaserror`	Spécifie que tous les avertissements doivent être traités comme des erreurs.
`vbc /warnaserror:42024`	Spécifie que cet avertissement 42024 doit être traité comme une erreur.
`vbc /warnaserror:42024,42025`	Spécifie que les avertissements 42024 et 42025 doivent être traités comme des erreurs.
`vbc /nowarn`	Spécifie qu'aucun avertissement ne doit être signalé.
`vbc /nowarn:42024`	Spécifie que cet avertissement 42024 ne doit pas être signalé.
`vbc /nowarn:42024,42025`	Spécifie que les avertissements 42024 et 42025 ne doivent pas être signalés.

Types d'avertissements

La liste suivante est une liste d'avertissements que vous pouvez traiter comme des erreurs.

Avertissement de conversion implicite

Généré pour les instances de conversion implicite. Elles ne concernent pas les conversions implicites d'un type numérique intrinsèque en une chaîne lors de l'utilisation de l'opérateur **&**. La valeur par défaut pour les nouveaux projets est désactivée.

ID : 42016

Avertissement d'appel de méthode à liaison tardive et de résolution de surcharge

Généré pour les instances de liaison tardive. La valeur par défaut pour les nouveaux projets est désactivée.

ID : 42017

Avertissements d'opérandes de type Objet

Générées lorsque les opérandes de type **Object** se produisent et créent une erreur avec **Option Strict On**. La valeur par défaut pour les nouveaux projets est activée.

ID : 42018 et 42019

Avertissements Les déclarations requièrent la clause 'As'

Générés lorsqu'une déclaration de variable, de fonction ou de propriété n'ayant pas une clause **As** aurait créé une erreur avec **Option Strict On**. Les variables qui n'ont pas de type qui leur est assigné sont supposées être de type **Object**. La valeur par défaut pour les nouveaux projets est activée.

ID : 42020 (déclaration de variable), 42021 (déclaration de fonction) et 42022 (déclaration de propriété).

Avertissements d'exception de référence null

Générés lorsqu'une variable est utilisée avant qu'une valeur ne lui ait été assignée. La valeur par défaut pour les nouveaux projets est activée.

ID : 42104, 42030

Avertissement de variable locale non utilisée

Généré lorsqu'une variable locale est déclarée mais jamais référencée. La valeur par défaut est activée.

ID : 42024

Avertissement d'accès de membre partagé via la variable d'instance

Généré lorsque l'accès à un membre partagé via une instance peut avoir des effets secondaires, ou lorsque l'accès à un membre partagé via une variable d'instance n'est pas la partie droite d'une expression ou est passé comme un paramètre. La valeur par défaut pour les nouveaux projets est activée.

ID : 42025

Avertissements d'opérateur récursif ou d'accès de propriété

Générés lorsque le corps d'une routine utilise le même opérateur ou propriété dans lequel il est défini. La valeur par défaut pour les nouveaux projets est activée.

ID: 42004 (opérateur), 42026 (propriété)

Avertissement de fonction ou d'opérateur sans valeur de retour

Généré lorsque la fonction ou l'opérateur n'a pas de valeur de retour spécifiée. Cela inclut l'omission d'un **Set** sur la variable locale implicite portant le même nom que la fonction. La valeur par défaut pour les nouveaux projets est activée.

ID: 42105 (fonction), 42016 (opérateur)

Avertissement de modificateur Overloads utilisé dans un module

Généré lorsque **Overloads** est utilisé dans un **Module**. La valeur par défaut pour les nouveaux projets est activée.

ID : 42028

Avertissements de blocs dupliqués ou superposés

Générés lorsqu'un bloc **Catch** n'est jamais atteint en raison de sa relation à d'autres blocs **Catch** qui ont été définis. La valeur par défaut pour les nouveaux projets est activée.

ID : 42029, 42031

Voir aussi

Tâches

Comment : activer ou désactiver les messages d'avertissement du compilateur

Référence

Assistant Exception, boîte de dialogue
/nowarn
/warnaserror (Visual Basic)
Compiler Warnings That Are Off by Default

Concepts

Types d'erreurs

Autres ressources

Tâches de gestion des exceptions

Gestion structurée des exceptions en Visual Basic

La gestion structurée des exceptions désigne la création d'un mécanisme de gestion des exceptions à l'aide d'une structure de contrôle contenant des exceptions, des blocs de code isolés et des filtres. Avec cette gestion structurée, votre code peut faire la différence entre plusieurs types d'erreurs et réagir en conséquence.

Dans cette section

Tâches de gestion des exceptions

Répertorie les tâches courantes de gestion des exceptions, par exemple comment intercepter une exception et comment filtrer des exceptions dans un bloc **Catch**.

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Fournit une vue d'ensemble des avantages et des structures de la gestion structurée des exceptions.

Classe d'exception dans Visual Basic

Décrit la classe Exception.

Procédure pas à pas : gestion structurée des exceptions

Fournit une procédure pas à pas pour créer une application simple et insérer un code de gestion des exceptions.

Résolution des problèmes liés à la gestion des exceptions

Répertorie les erreurs couramment rencontrées lors de l'utilisation de la gestion structurée des exceptions et explique comment les corriger.

Référence

Throw, instruction (Visual Basic)

Décrit l'instruction **Throw** et explique comment l'utiliser pour lever une exception.

Try...Catch...Finally, instruction (Visual Basic)

Décrit les instructions **Try...Catch...Finally** et explique comment les utiliser pour tester des blocs de code.

Rubriques connexes

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Décrit les avantages de la gestion structurée des exceptions et ceux de la gestion non structurée.

Vue d'ensemble de la gestion non structurée des exceptions

Décrit la gestion non structurée des exceptions et explique comment l'utiliser.

Tâches de gestion des exceptions

Visual Basic prend en charge la gestion structurée et non structurée des exceptions (erreurs). En insérant du code destiné à gérer les exceptions spécifiques dans votre application, vous pouvez gérer la plupart des erreurs que les utilisateurs sont susceptibles de rencontrer. Utilisez la gestion structurée et non structurée des exceptions pour empêcher les erreurs potentielles d'interférer avec votre application.

Dans cette section

Comment : intercepter une exception dans Visual Basic

Explique comment utiliser un bloc **Try...Catch** pour intercepter une exception.

Comment : lever une exception dans Visual Basic

Explique comment utiliser l'instruction **Throw** pour lever une exception.

Comment : implémenter des blocs Try...Catch d'E/S dans Visual Basic

Explique comment utiliser un bloc **Try...Catch** pour tester une opération d'entrée ou de sortie.

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Montre comment utiliser un bloc **Try...Catch** pour tester une section de code.

Comment : nettoyer des ressources avec un bloc Try...Finally dans Visual Basic

Explique comment utiliser un bloc **Try...Catch...Finally** pour nettoyer des ressources utilisées dans le code.

Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic

Explique comment filtrer les types d'erreurs spécifiques dans un bloc **Try...Catch**.

Comment : afficher le message d'une exception dans Visual Basic

Explique comment récupérer la chaîne associée à un message d'erreur.

Comment : créer des classes d'exception dans Visual Basic

Explique comment créer une nouvelle classe d'exception.

Comment : vérifier l'exception interne d'une exception

Explique comment déterminer l'exception d'origine en examinant la propriété **InnerException** d'une exception.

Référence

Throw, instruction (Visual Basic)

Décrit l'instruction **Throw** et son utilisation.

Try...Catch...Finally, instruction (Visual Basic)

Décrit les instructions **Try**, **Catch** et **Finally**, ainsi que leur utilisation.

Comment : intercepter une exception dans Visual Basic

Cet exemple illustre comment utiliser un bloc **Try** et **Catch** pour intercepter des exceptions.

Exemple

Cet exemple illustre comment utiliser un bloc **Try...Catch** pour intercepter une exception OverflowException.

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
Dim Top As Double = 5
Dim Bottom As Double = 0
Dim Result As Integer
Try
    Result = CType(Top / Bottom, Integer)
Catch Exc As System.OverflowException
    MsgBox("Attempt to divide by zero resulted in overflow")
End Try
```

Cet exemple nécessite :

- Référence à l'espace de noms System.

L'exemple de code suivant implémente un bloc **Try...Catch** qui gère Exception, IOException et toutes les exceptions qui dérivent de **IOException**.

VB

```
Try
    ' Add code for your I/O task here.
Catch dirNotFound As System.IO.DirectoryNotFoundException
    Throw dirNotFound
Catch fileNotFound As System.IO.FileNotFoundException
    Throw fileNotFound
Catch pathTooLong As System.IO.PathTooLongException
    Throw pathTooLong
Catch ioEx As System.IO.IOException
    Throw ioEx
Catch security As System.Security.SecurityException
    Throw security
Catch ex As Exception
    Throw ex
Finally
    ' Dispose of any resources you used or opened in the Try block.
End Try
```

Ajoutez le code à exécuter au bloc **Try**.

Programmation fiable

Utilisez ce bloc de code comme point de départ pour encapsuler une opération de données dans une instruction **Try...Catch**. Ce bloc **Try...Catch** est conçu pour intercepter et lever de nouveau toutes les exceptions. Il se peut que cela ne soit pas le bon choix pour votre projet. Pour une discussion sur les options de gestion des exceptions, consultez Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense.

Vous pouvez réduire la probabilité des exceptions en utilisant des contrôles Windows Forms, tels que le composant OpenFileDialog, composant (Windows Forms) et les contrôles de composant SaveFileDialog, composant (Windows Forms) qui limite les choix de l'utilisateur aux noms de fichiers valides. La propriété FileInfo.Exists peut vérifier si un fichier existe avant d'essayer de l'ouvrir. Notez toutefois que l'utilisation de ces contrôles et de ces classes n'est pas à toute épreuve. Le système de fichiers peut changer entre le moment où l'utilisateur sélectionne un fichier et celui où le code s'exécute. La gestion des

exceptions sera par conséquent presque toujours nécessaire lors de l'utilisation de fichiers.

Sécurité

Pour de nombreuses tâches sur les fichiers, l'assembly requiert un niveau de privilège accordé par la classe FileIOPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code. L'utilisateur nécessite également un accès au fichier, s'il existe. Pour plus d'informations, consultez Listes de contrôle d'accès.

Ne vous basez pas sur l'extension d'un nom de fichier pour en déterminer le contenu. Par exemple, il se peut que le fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Voir aussi

Tâches

Comment : lever une exception dans Visual Basic
Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic
Comment : nettoyer des ressources avec un bloc Try...Finally dans Visual Basic
Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic
Comment : vérifier l'exception interne d'une exception
Résolution des problèmes liés à la gestion des exceptions

Concepts

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Comment : lever une exception dans Visual Basic

Cet exemple lève une exception ApplicationException.

Exemple

VB

```
Throw New ApplicationException
' Code to react to possible causes of the exception.
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez
Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite :

- Référence à l'espace de noms System.

Programmation fiable

Les exceptions non gérées peuvent interrompre l'exécution de l'application. Utilisez des blocs **Try...Catch** pour anticiper et gérer des exceptions possibles.

Voir aussi

Tâches

Comment : intercepter une exception dans Visual Basic
Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic
Comment : nettoyer des ressources avec un bloc Try...Finally dans Visual Basic
Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic
Comment : vérifier l'exception interne d'une exception
Résolution des problèmes liés à la gestion des exceptions

Référence

Exception Class
Throw, instruction (Visual Basic)

Concepts

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Comment : implémenter des blocs Try...Catch d'E/S dans Visual Basic

L'exemple de code suivant implémente un bloc **Try...Catch** qui traite Exception, IOException et toutes les exceptions qui dérivent de **IOException**.

Exemple

Cet exemple va du plus spécifique au moins spécifique ; chaque exception est testée tour à tour.

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il est localisé dans **Système de fichiers - Traitement de lecteurs, de dossiers et de fichiers**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
Try
    ' Add code for your I/O task here.
Catch dirNotFound As System.IO.DirectoryNotFoundException
    ' Code to handle DirectoryNotFoundException.
Catch fileNotFound As System.IO.FileNotFoundException
    ' Code to handle FileNotFoundException.
Catch pathTooLong As System.IO.PathTooLongException
    ' Code to handle PathTooLongException.
Catch ioEx As System.IO.IOException
    ' Code to handle IOException.
Catch security As System.Security.SecurityException
    ' Code to handle SecurityException.
Catch ex As Exception
    ' Rethrow exception if anything else has occurred.
    Throw ex
Finally
    ' Dispose of any resources you used or opened in the Try block.
End Try
```

Compilation du code

Ajoutez le code que vous souhaitez exécuter au bloc **Try**.

Programmation fiable

Utilisez ce bloc de code comme point de départ pour encapsuler une opération de données dans une instruction **Try...Catch**. Ce bloc **Try...Catch** est conçu pour intercepter et lever de nouveau toutes les exceptions. Ce n'est pas forcément la meilleure solution pour votre projet. Pour obtenir une description des options de gestion des exceptions, consultez Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense.

Vous pouvez réduire la probabilité d'exceptions en utilisant les contrôles Windows Forms, tels que le composant OpenFileDialog, composant (Windows Forms) et les contrôles du composant SaveFileDialog, composant (Windows Forms) qui limitent les choix de l'utilisateur aux noms de fichiers valides. La propriété FileInfo.Exists peut vérifier si un fichier existe avant d'essayer de l'ouvrir. Toutefois, l'utilisation de ces contrôles et de ces classes n'est pas sans poser de problèmes. Le système de fichiers peut changer entre le moment où l'utilisateur sélectionne un fichier et le moment où le code s'exécute. La gestion des exceptions est donc presque toujours requise lors de l'utilisation de fichiers.

Sécurité

Pour de nombreuses tâches liées aux fichiers, l'assembly requiert un niveau de privilège accordé par la classe UIPermission. Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code. L'utilisateur doit avoir également accès au fichier, s'il existe. Pour plus d'informations, consultez Listes de contrôle d'accès.

Ne vous basez pas sur l'extension du nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut que le fichier nommé Form1.vb ne soit pas un fichier source Visual Basic.

Voir aussi

Tâches

Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic

Comment : vérifier l'exception interne d'une exception

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Méthodes conseillées pour l'utilisation d'extraits de code IntelliSense

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Une instruction **Catch** peut être utilisée dans un bloc **Try** afin d'intercepter et de répondre à une exception spécifique ou à plusieurs exceptions. Si une exception se produit lors de l'exécution du code contenu dans la section **Try**, le compilateur Visual Basic examine chaque instruction **Catch** du bloc jusqu'à ce qu'il en trouve une dont la condition correspond à l'exception. Si la recherche est infructueuse, une erreur se produit.

Pour intercepter une exception spécifique

1. Utilisez un bloc **Try** pour tester le bloc de code, en l'englobant dans **Try** et **End Try**, comme dans l'exemple suivant, qui copie le fichier `MyLog` vers le même répertoire et le renomme `BackupLog`.

   VB

   ```
   Try
       My.Computer.FileSystem.CopyFile("MyLog", "BackupLog")
   Catch ex As System.IO.IOException
       MsgBox("An error occurred")
   End Try
   ```

2. Dans le bloc **Try**, fournissez des instructions **Catch** visant des types spécifiques d'erreurs, en allant du plus spécifique au moins spécifique. Ici, l'instruction **Catch** intercepte d'abord toutes les exceptions IOException avant de rechercher des exceptions générales.

   VB

   ```
   Catch ex As System.IO.FileNotFoundException
       MsgBox("No such file in this directory.")
   Catch ex As System.Exception
       MsgBox("An unspecified error occurred.")
   ```

Voir aussi

Tâches

Comment : nettoyer des ressources avec un bloc Try...Finally dans Visual Basic
Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic
Comment : vérifier l'exception interne d'une exception
Résolution des problèmes liés à la gestion des exceptions

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Comment : nettoyer des ressources avec un bloc Try...Finally dans Visual Basic

Une instruction **Finally** peut être utilisée dans un bloc **Try** afin de s'assurer que les ressources allouées sont nettoyées. Le code dans un bloc **Finally** s'exécute après le code de gestion des exceptions, mais avant que le contrôle ne revienne à la procédure appelante. Le code dans un bloc **Finally** s'exécutera même si votre code lève une exception et même si vous ajoutez une instruction **Exit Function** (ou **Exit Sub**) explicite dans un bloc **Catch**.

Si vous n'avez pas besoin d'intercepter des exceptions spécifiques, l'instruction **Using** se comporte comme un bloc **Try... Finally** et garantit l'élimination des ressources, quelle que soit la manière dont vous quittez le bloc. Cela est vrai même dans le cas d'une exception non gérée. Pour plus d'informations, consultez Using, instruction (Visual Basic).

Pour nettoyer des ressources avec une instruction Finally

- Placez le code que vous souhaitez exécuter indépendamment des exceptions dans le bloc **Finally**. Le code suivant crée un StreamReader et l'utilise pour lire à partir d'un fichier.

  VB

```
Dim reader As New System.IO.StreamReader("C:\testfile")
Try
    reader.ReadToEnd()
Catch ex As System.IO.IOException
    MsgBox("Could not read file")
Finally
    'This command is executed whether or not the file can be read
    reader.Close()
End Try
```

Voir aussi

Tâches

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Comment : vérifier l'exception interne d'une exception

Comment : supprimer une ressource système

Référence

Using, instruction (Visual Basic)

Concepts

Gestion des exceptions pour les utilisateurs de Visual Basic 6.0

Autres ressources

Tâches de gestion des exceptions

Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic

Les instructions **Catch** fournissent plusieurs options pour filtrer des erreurs. L'une des méthodes consiste à effectuer un filtrage par type d'exception. Dans de tels cas, il est important de se déplacer du type d'exception le plus spécifique au plus général, puisque les instructions **Catch** sont exécutées dans l'ordre.

Une clause **When** peut également être utilisée pour filtrer sur une expression conditionnelle, telle qu'un numéro d'erreur spécifique. Vous pouvez aussi combiner ces deux approches.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide est orientée sur les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour filtrer en fonction du type d'exception

- Insérez une instruction **Catch** pour chaque type d'exception à vérifier, en allant du plus spécifique au plus général.

  VB

```
Try
    Throw New Exception
Catch ex As System.IO.IOException
    ' Code reacting to IOException
Catch ex As System.NullReferenceException
    ' Code reacting to NullReferenceException
Catch ex As Exception
    ' Code reacting to any exception
End Try
```

Pour filtrer sur une expression conditionnelle

- Utilisez une instruction **Catch When** pour filtrer sur une expression conditionnelle. Si l'expression conditionnelle est évaluée comme **True**, le code qui suit le bloc **Catch** sera exécuté.

  VB

```
Try
    ' Code goes here.
    ' Check for type mismatch error.
Catch ex As Exception When Err.Number = 5
    ' Code reacting to exception.
End Try
```

Voir aussi

Tâches

Comment : vérifier l'exception interne d'une exception
Résolution des problèmes liés à la gestion des exceptions

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Gestion des exceptions pour les utilisateurs de Visual Basic 6.0
Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Comment : afficher le message d'une exception dans Visual Basic

L'objet exception inclut plusieurs propriétés qui aident à identifier l'emplacement du code, le type, l'URL Helpfile et la cause de l'exception. L'une d'elles, la propriété **Message**, décrit l'exception actuelle.

Pour afficher la chaîne associée à une exception

- Utilisez la propriété **Message** pour afficher des informations sur l'exception actuelle. Cet exemple intercepte une exception WebException et affiche le message associé.

  VB

```
Try
    Throw New System.Net.WebException
Catch ex As System.Net.WebException
    MsgBox("An exception occurred:" & vbCrLf & ex.Message)
End Try
```

Voir aussi

Tâches

Comment : intercepter une exception dans Visual Basic
Comment : lever une exception dans Visual Basic
Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic
Comment : vérifier l'exception interne d'une exception
Résolution des problèmes liés à la gestion des exceptions

Référence

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Gestion structurée des exceptions en Visual Basic

Comment : créer des classes d'exception dans Visual Basic

Vous pouvez créer vos propres classes d'exception d'application à partir d'un héritage de la classe Exception. Suivez les méthodes de codage conseillées qui consistent à terminer le nom de classe de votre exception par le mot `Exception`, tel que `OutOfMoneyException` ou `TooMuchRainException`.

L'exemple de code suivant fournit une implémentation de base d'une classe d'exception.

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
Public Class YourProblemException
    Inherits Exception

    Public Sub New()
        ' Add other code for custom properties here.
    End Sub

    Public Sub New(ByVal message As String)
        MyBase.New(message)
        ' Add other code for custom properties here.
    End Sub

    Public Sub New(ByVal message As String, ByVal inner As Exception)
        MyBase.New(message, inner)
        ' Add other code for custom properties here.
    End Sub

    Public Sub New( _
        ByVal info As System.Runtime.Serialization.SerializationInfo, _
        ByVal context As System.Runtime.Serialization.StreamingContext)
        MyBase.New(info, context)
        ' Insert code here for custom properties here.
    End Sub
End Class
```

Compilation du code

- Remplacez `YourProblemException` par le nom de la classe d'exception que vous souhaitez créer. En général, les noms de classes d'exception se terminent par « Exception ». Ajoutez des propriétés pour transmettre des informations supplémentaires relatives à l'erreur qui s'est produite.

Sécurité

Ne divulguez pas d'informations sur l'application ou ses données au cours de la gestion d'une exception. Ces informations pourraient être utilisées pour attaquer votre application.

Voir aussi

Tâches

Résolution des problèmes liés à la gestion des exceptions

Référence

ApplicationException Class

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Gestion des exceptions pour les utilisateurs de Visual Basic 6.0

Autres ressources

Tâches de gestion des exceptions

Comment : vérifier l'exception interne d'une exception

Dans les cas où une exception est levée comme conséquence directe d'une exception antérieure, la propriété InnerException décrit l'erreur d'origine. Ces informations peuvent vous permettre de gérer l'erreur plus efficacement. S'il n'y a aucune erreur d'origine, la valeur de **InnerException** sera une référence Null ou **Nothing** dans Visual Basic. Cette propriété est en lecture seule.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide est orientée sur les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour vérifier une InnerException

- Vérifiez la propriété **InnerException** de l'exception pour déterminer la cause de l'erreur d'origine.

  VB

```
Try
    My.Computer.FileSystem.CopyFile("file1", "file2")
Catch ex As System.IO.IOException
    MsgBox(ex.InnerException)
End Try
```

Voir aussi

Tâches

Comment : afficher le message d'une exception dans Visual Basic

Résolution des problèmes liés à la gestion des exceptions

Référence

InnerException

Autres ressources

Tâches de gestion des exceptions

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Visual Basic prend en charge la gestion structurée des exceptions, qui vous permet de créer et de gérer des programmes à l'aide de gestionnaires d'erreurs robustes et complets. La gestion structurée des exceptions est conçue sous forme de code pour détecter et répondre aux erreurs d'exécution en associant une structure de contrôle (semblable à **Select Case** ou **While**) à des exceptions, des blocs de code protégés et des filtres.

L'instruction **Try...Catch...Finally** vous permet de protéger des blocs de code qui présentent le risque de générer des erreurs. Vous pouvez imbriquer les gestionnaires d'exceptions ; en outre, les variables déclarées dans chaque bloc auront une portée locale.

Try...Catch...Finally, instruction

Le code suivant illustre la structure d'une instruction **Try...Catch...Finally**.

```
Try
' Starts a structured exception handler.
' Place executable statements that may generate
' an exception in this block.
Catch [optional filters]
' This code runs if the statements listed in
' the Try block fail and the filter on the Catch statement is true.
[Additional Catch blocks]
Finally
' This code always runs immediately before
' the Try statement exits.
End Try
' Ends a structured exception handler.
```

Le bloc **Try** d'un gestionnaire d'exceptions **Try...Catch...Finally** contient la section de code dans laquelle il faut contrôler les exceptions. Si une erreur se produit durant l'exécution de cette section, Visual Basic examine chaque instruction **Catch** du bloc **Try...Catch...Finally** jusqu'à ce qu'il en trouve une correspondant à l'erreur. Si une instruction est trouvée, le contrôle est transféré vers la première ligne de code du bloc **Catch**. Si aucune instruction **Catch** correspondante n'est trouvée, la recherche se poursuit sur les instructions **Catch** du bloc **Try...Catch...Finally** extérieur qui contient le bloc dans lequel l'exception s'est produite. Ce processus se poursuit dans la pile entière jusqu'à ce qu'un bloc **Catch** correspondant soit trouvé dans la procédure en cours. Si la recherche est infructueuse, une erreur se produit.

Le code de la section **Finally** s'exécute toujours en dernier, juste avant la fin de la portée du bloc de gestion d'erreur, indépendamment du fait que le code des blocs **Catch** se soit exécuté ou non. Placez le code de nettoyage, par exemple pour fermer des fichiers et libérer des objets, dans la section **Finally**. Si vous n'êtes pas obligé d'intercepter les exceptions, mais que vous devez nettoyer les ressources, envisagez l'utilisation de l'instruction **Using** plutôt que d'une section **Finally**. Pour plus d'informations, consultez Using, instruction (Visual Basic).

Filtrage des erreurs dans le bloc Catch

Les blocs **Catch** permettent l'utilisation de trois options pour filtrer les erreurs spécifiques. Dans un cas, les erreurs sont filtrées selon la classe de l'exception (ici `ClassLoadException`), comme indiqué dans le code suivant.

```
Try
' "Try" block.
Catch e as ClassLoadException
' "Catch" block.
Finally
' "Finally" block.
End Try
```

Si une erreur `ClassLoadException` se produit, le code contenu dans le bloc **Catch** spécifié s'exécute.

Dans le deuxième cas de filtrage des erreurs, la section **Catch** peut filtrer toutes les expressions conditionnelles. L'utilisation la plus courante de ce type de filtre **Catch** permet de tester des numéros d'erreur spécifiques, comme indiqué dans le code suivant.

```
Try
   ' "Try" block.
Catch When ErrNum = 5 'Type mismatch.
   ' "Catch" block.
Finally
   ' "Finally" block.
End Try
```

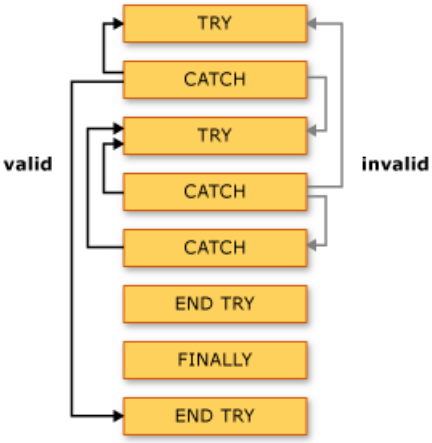
Lorsque Visual Basic trouve le gestionnaire d'erreur correspondant, il exécute le code contenu dans ce gestionnaire, puis passe le contrôle au bloc **Finally**.

Remarque
Lors d'une tentative de recherche d'un bloc **Catch** pour gérer une exception, chaque gestionnaire de bloc est évalué jusqu'à l a découverte d'une correspondance. Puisque ces gestionnaires peuvent être des appels à des fonctions, des effets secondaire s inattendus peuvent se produire ; par exemple, un appel de ce type pourrait modifier une variable publique qui est ensuite u tilisée dans le code d'un bloc **Catch** différent qui finit par gérer l'exception.

Une troisième alternative consiste à associer la première et la deuxième option et à utiliser ces deux options pour la gestion des exceptions. Vos instructions **Catch** doivent aller de la plus spécifique à la moins spécifique. Un bloc **Catch** par lui-même interceptera toutes les exceptions dérivées de Exception et par conséquent doit toujours être le dernier bloc avant **Finally**.

Branchement en dehors de blocs Try...Catch

Il est possible de créer une branche à partir d'un bloc **Catch** vers l'instruction **Try** initiale ou l'instruction **End Try**, mais il n'est pas possible de créer une branche vers un bloc **Try...Catch** englobant. En voici une illustration :



Exemple de gestionnaire d'exceptions structuré

L'exemple suivant illustre un autre gestionnaire d'erreur de base utilisant l'instruction **Try...Catch...Finally**.

```
Function GetStringsFromFile(ByVal FileName As String) As Collection
Dim Strings As New Collection
Dim Stream As System.IO.StreamReader = System.IO.File.OpenText(FileName)   'Open the file.

Try
While True
' Loop terminates with an EndOfStreamException
' error when end of stream is reached.
Strings.Add(Stream.ReadLine())
End While
Catch eos As System.IO.EndOfStreamException
' No action is necessary; end of stream has been reached.
Catch IOExcep As System.IO.IOException
' Some kind of error occurred. Report error and clear collection.
MsgBox(IOExcep.Message)
Strings = Nothing
Finally
Stream.Close()
' Close the file.
```

```
    End Try

    Return Strings
    End Function
```

Le bloc **Finally** est toujours exécuté, quelle que soit l'action qui se produit dans les blocs **Catch** précédents. Vous ne pouvez pas utiliser **Resume** ni **Resume Next** dans la gestion structurée des exceptions.

Remarque
Dans l'exemple précédent, toute exception autre que la classe **IOException** ou la classe EndOfStreamException est retournée non gérée à l'appelant.

Voir aussi

Tâches

Résolution des problèmes liés à la gestion des exceptions

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Introduction à la gestion des exceptions

Types d'erreurs

Vue d'ensemble de la gestion non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Classe d'exception dans Visual Basic

Pour faciliter l'utilisation de la gestion structurée des exceptions, Visual Basic permet de séparer le code standard du code de gestion des exceptions. Le code de gestion des exceptions accède à une instance de la classe Exception, ce qui vous permet d'extraire les informations relatives aux exceptions rencontrées.

Notes

Lorsqu'une exception est levée, l'objet **Err** global est défini et une nouvelle instance d'une classe **Exception** est créée.

Les propriétés de la classe **Exception** aident à identifier l'emplacement dans le code, le type et la cause des exceptions. Par exemple, la propriété StackTrace répertorie les méthodes appelées qui ont abouti à l'exception, ce qui vous permet de rechercher l'emplacement de l'erreur dans le code. La propriété Message retourne un message décrivant l'erreur ; vous pouvez le modifier pour le rendre plus facile à comprendre. Si vous ne fournissez pas de chaîne de texte de message d'erreur, la valeur par défaut est utilisée. HelpLink obtient ou définit un lien vers un fichier d'aide associé. Source obtient ou définit une chaîne contenant le nom de l'objet qui provoque l'erreur ou le nom de l'assembly d'où provient l'exception.

Remarque
Évitez d'écrire du code qui teste des messages d'erreur spécifiques, car ces messages peuvent varier entre les versions de la classe dont ils sont issus. Testez plutôt le type d'exception ou utilisez le numéro d'exception à partir de l'objet **Err** global.

Propriétés de classe d'exception

Le tableau suivant répertorie et décrit les propriétés de la classe **Exception**.

Propriété	Description
HelpLink	Obtient ou définit un lien vers le fichier d'aide associé à cette exception.
HResult	Obtient ou définit `HRESULT`, une valeur numérique codée qui est assignée à une exception spécifique.
InnerException	Obtient l'instance **Exception** qui a provoqué l'exception actuelle.
Message	Obtient un message qui décrit l'exception actuelle.
Source	Obtient ou définit le nom de l'application ou objet qui a provoqué l'erreur.
StackTrace	Obtient la représentation sous forme de chaîne des frames de la pile des appels au moment où l'exception actuelle a été levée.
TargetSite	Obtient la méthode qui lève l'exception actuelle.

Voir aussi

Tâches

Résolution des problèmes liés à la gestion des exceptions

Concepts

Introduction à la gestion des exceptions

Types d'erreurs

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Vue d'ensemble de la gestion non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Procédure pas à pas : gestion structurée des exceptions

Si vous pouvez toujours employer l'instruction **On Error** pour traiter des exceptions dans votre code et permettre la gestion non structurée des exceptions, Visual Basic 2005 prend également en charge la gestion structurée des exceptions, ce qui vous permet de créer et de gérer des programmes avec une gestion d'erreurs complète. Dans la gestion structurée des exceptions, des blocs de code testent des situations spécifiques et réagissent en conséquence.

Cette procédure pas à pas montre comment ajouter une gestion structurée des exceptions à un programme. Elle explique en particulier comment utiliser l'instruction **Try...Catch...Finally** pour gérer les exceptions et comment effectuer le filtrage des erreurs dans les blocs **Catch**.

Remarque
Vous ne pouvez pas combiner à la fois la gestion structurée et non structurée des exceptions dans une seule procédure.

Remarque
Les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide sel on vos paramètres actifs ou votre édition. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Création de l'application

L'application suivante est un formulaire de commande d'un client pour une société qui vend des ours en peluche. L'interface utilisateur se compose des contrôles suivants :

- Un contrôle TextBox pour le nom du client.
- Deux contrôles ComboBox pour sélectionner la couleur et la taille de l'ours.
- Un bouton de commande Button.
- Trois étiquettes qui expliquent la signification de chaque contrôle à l'utilisateur.
- Lorsque l'utilisateur entre les informations demandées, puis clique sur le bouton Commander, l'application affiche un résumé de la commande.

Pour créer l'application

1. Dans le menu **Fichier**, choisissez **Nouveau projet**. La boîte de dialogue **Nouveau projet** s'affiche.
2. Dans la fenêtre **Types de projets**, sélectionnez **Visual Basic et Windows** si ce n'est déjà fait, sélectionnez **Application Windows** dans la fenêtre **Modèles**.
3. Dans la fenêtre **Propriétés** sous **Nom**, entrez **TeddyBearProject**, puis cliquez sur **OK**. Le projet est ajouté à l'Explorateur de solutions et le Concepteur Windows Forms s'ouvre.
4. Ajoutez les contrôles au formulaire et définissez leurs propriétés comme spécifié.

Contrôle	**Propriétés**	**Valeurs de la propriété**
Étiquette	Nom Texte	customerLabel Formulaire Commande Ours
Zone de texte	Nom Texte	customerName Nom du client
Étiquette	Nom Texte	bearColorLabel Couleurs disponibles

Liste déroulante	Nom Éléments Texte	bearColor Noir, brun, tacheté Couleur de l'ours
Étiquette	Nom Texte	bearSizeLabel Tailles disponibles
Liste déroulante	Nom Éléments Texte	bearSize Petit, normal, grand Taille
Bouton	Nom Texte	order Commande

Ajout de fonctionnalités

Maintenant que les contrôles et leurs propriétés sont ajoutés et définis, vous devez insérer du code derrière pour qu'ils fonctionnent. Le code suivant traite l'événement Click pour le bouton Commander et affiche un message pour le client.

Pour ajouter des fonctionnalités au formulaire et à ses contrôles

- Ajoutez le code suivant à l'événement **Click** du bouton Commander.

  VB

```
Dim bearOrder As String
bearOrder = _
  String.Format("You have ordered a {0} {1} bear.", _
  bearSize.SelectedItem, bearColor.SelectedItem)
MsgBox(bearOrder)
```

Vous pouvez désormais ajouter un code de gestion structurée des exceptions à l'application.

Ajout d'un bloc Try...Catch

Pour vous assurer que le client a indiqué une couleur pour l'ours, ajoutez une instruction **Try...Catch**. N'oubliez pas :

- Une clause **Catch** sans identificateur intercepte toutes les exceptions.
- Une clause **Catch** avec une clause **When** n'intercepte les exceptions que lorsque l'expression correspond à la valeur **True** ; le type de l'expression doit être implicitement convertible en type **Boolean**.

Pour ajouter un bloc Try...Catch simple

- Ajoutez le code suivant à l'événement **Click** du bouton Commander après la section qui teste les valeurs de couleur et de taille, c'est-à-dire après `String.Format ("You have ordered a {0} {1} bear.", BearSize.SelectedItem, BearColor.SelectedItem)`. Ce code lève une exception si une valeur de couleur non valide est trouvée. Lors de l'ajout des instructions **Try**, notez que l'éditeur fournit automatiquement **End Try** à la fin de l'instruction.

  VB

```
Try
    If ((bearColor.SelectedIndex < 0) Or _
          (bearColor.SelectedIndex > 2)) Then
        Throw New System.Exception()
    End If
    ' The Catch statement handles errors caused by a lack of bear color.
Catch ex As Exception When bearColor.SelectedIndex < 0
```

```
        bearOrder = String.Format("You must select a bear color!")
    Finally
        Beep() ' Beep at the end.
    End Try
```

Pour ajouter une autre clause Catch

1. Ajoutez un nouvel élément (« Violet ») au contrôle ComboBox `bearColor`.
2. Ajoutez le code suivant après la ligne de code `bearOrder = String.Format("You must select a bear color!")`.

   VB

```
    Catch ex As Exception When bearColor.SelectedIndex = 3
        bearOrder = String.Format("There are no bears of that color.")
```

Vous pouvez ajouter autant de clauses **Catch** que nécessaire à votre code.

Test

Vous pouvez à présent tester l'application afin de vous assurer qu'elle fonctionne correctement.

Pour générer et exécuter l'application

1. Dans le menu **Générer**, sélectionnez **Générer TeddyBearProject**.
2. Appuyez sur F5 pour exécuter l'application. Le formulaire principal s'affiche.

Pour tester l'application

1. Entrez un nom dans la zone de texte **Nom du client**, puis sélectionnez la couleur et la taille de l'ours dans les listes déroulantes **Couleurs disponibles** et **Tailles disponibles**.
2. Cliquez sur le bouton **Ordre**. Si vous avez sélectionné la couleur **Noir**, **Brun** ou **Tacheté** à l'étape précédente, un message apparaît indiquant la taille et la couleur de l'ours que vous avez commandé.

   Si vous n'avez pas spécifié de couleur, un message vous invite à en sélectionner une.
3. Cliquez sur **OK** pour fermer le message.
4. Dans la zone de texte **Couleurs disponibles**, sélectionnez **Violet**, puis cliquez sur le bouton **Ordre**. Un message apparaît indiquant que les ours violets ne sont pas disponibles.

Voir aussi

Référence

On Error, instruction (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)
Throw, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Autres ressources

Tâches de gestion des exceptions

Résolution des problèmes liés à la gestion des exceptions

Cette rubrique décrit les problèmes couramment rencontrés lors de l'utilisation des exceptions et comment les résoudre.

Visual Basic prend en charge la gestion structurée des exceptions, qui vous permet de créer et de gérer des programmes à l'aide de gestionnaires d'erreurs robustes et complets. La gestion structurée des exceptions est conçue sous forme de code pour détecter et répondre aux erreurs d'exécution en associant une structure de contrôle (semblable à **Select Case** ou **While**) à des exceptions, des blocs de code protégés et des filtres.

Exceptions internes

Dans les cas où une exception est levée comme conséquence directe d'une exception antérieure, la propriété InnerException décrit l'erreur d'origine. Ces informations vous permettent de gérer l'erreur plus efficacement. S'il n'y a aucune erreur d'origine, la valeur de **InnerException** sera une référence Null ou **Nothing** en Visual Basic. Cette propriété est en lecture seule. Pour plus d'informations, consultez Comment : vérifier l'exception interne d'une exception.

Instructions Try...Catch

Votre code ne peut pas intercepter correctement les exceptions si vous ordonnez vos blocs **Catch** de manière incorrecte. Vos instructions **Catch** doivent aller de la plus spécifique à la moins spécifique. Un bloc **Catch** par lui-même interceptera toutes les exceptions dérivées de Exception et par conséquent doit toujours être placé comme le dernier bloc avant **Finally**.

Voir aussi

Tâches

Procédure pas à pas : gestion structurée des exceptions

Concepts

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Autres ressources

Tâches de gestion des exceptions

Gestion non structurée des exceptions en Visual Basic

Les rubriques de cette section décrivent la gestion non structurée des exceptions, l'objet **Err** et la façon de récupérer des informations à partir d'erreurs.

Dans cette section

Vue d'ensemble de la gestion non structurée des exceptions

Décrit la gestion non structurée des exceptions et la façon d'utiliser l'objet **Err**.

Objet Err dans la gestion non structurée des exceptions

Décrit l'objet **Err** et ses propriétés.

Comment : extraire des informations à partir d'un objet Error

Montre comment obtenir des informations des propriétés d'un objet **Err**.

Erreurs récupérables dans Visual Basic

Répertorie les erreurs récupérables.

Référence

Err, objet (Visual Basic)

Décrit l'objet **Err** et répertorie ses membres.

Description, propriété (objet Err)

Décrit la propriété **Description** de l'objet **Err** et son utilisation.

HelpContext, propriété (objet Err)

Décrit la propriété **HelpContext** de l'objet **Err** et son utilisation.

HelpFile, propriété (objet Err)

Décrit la propriété **Helpfile** de l'objet **Err** et son utilisation.

LastDllError, propriété (objet Err)

Décrit la propriété **LastDLLError** de l'objet **Err** et son utilisation.

Number, propriété (objet Err)

Décrit la propriété **Number** de l'objet **Err** et son utilisation.

Source, propriété (objet Err)

Décrit la propriété **Source** de l'objet **Err** et son utilisation.

On Error, instruction (Visual Basic)

Décrit l'instruction **On Error** et son utilisation.

Resume, instruction

Décrit l'instruction **Resume** et son utilisation.

Exit, instruction (Visual Basic)

Décrit l'instruction **Exit** et son utilisation.

Rubriques connexes

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Décrit les avantages et les inconvénients de la gestion structurée des exceptions et ceux de la gestion non structurée.

Types d'erreurs

Répertorie les types d'erreurs rencontrées dans le code.

Vue d'ensemble de la gestion non structurée des exceptions

Dans la *gestion non structurée des exceptions*, vous devez placer une instruction **On Error** au début d'un bloc de code pour pouvoir gérer les erreurs qui se produisent dans ce bloc. Lorsqu'une exception est générée dans une procédure après l'exécution de l'instruction **On Error**, le programme effectue un branchement vers l'argument de ligne spécifié dans l'instruction **On Error**. L'argument de ligne, qui représente le numéro de ligne ou l'étiquette de ligne, indique l'emplacement du gestionnaire d'exceptions.

Parfois, un appel est effectué de la procédure d'origine vers une autre procédure, et une exception se produit dans la procédure appelée. Dans ces cas-là, si la procédure appelée ne gère pas l'exception, celle-ci est retournée à la procédure appelante, et l'exécution effectue un branchement vers l'argument de ligne.

Remarque

Une gestion non structurée des erreurs à l'aide de **On Error** peut nuire aux performances de l'application et produire un cod e difficile à déboguer et à gérer. La gestion structurée des erreurs est la méthode conseillée. Pour plus d'informations, consul tez Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

On Error GoTo Line

L'instruction **On Error GoTo Line** suppose que le code de gestion des erreurs débute à la ligne spécifiée dans l'argument *line* requis. Si une erreur d'exécution se produit, le contrôle effectue un branchement vers l'étiquette ou le numéro de ligne spécifié dans l'argument, ce qui entraîne l'activation du gestionnaire d'erreur. La ligne spécifiée doit se trouver dans la même procédure que l'instruction **On Error GoTo Line**, sinon Visual Basic génère une erreur de compilation. L'exemple suivant illustre l'utilisation d'un gestionnaire d'erreur avec une étiquette de ligne :

```
Sub TestSub
   On Error GoTo ErrorHandler
      ' Code that may or may not contain errors.
   Exit Sub

   ErrorHandler:
      ' Code that handles errors.
      Resume
End Sub
```

L'exemple contient un gestionnaire d'erreur nommé `ErrorHandler`. Si le code de la sous-routine `TestSub` génère une erreur, Visual Basic exécute immédiatement le code qui suit l'étiquette `ErrorHandler`. À la fin du bloc de gestion d'erreur, l'instruction **Resume** repasse le contrôle à la ligne de code où l'erreur s'est produite initialement. Le reste de la sous-routine continue alors à s'exécuter comme si l'erreur ne s'était pas produite.

Remarque

Vous devez placer une instruction **Exit Sub** immédiatement avant le bloc de gestion d'erreur. Sinon, Visual Basic exécute le c ode de gestion d'erreur lorsqu'il atteint la fin de la sous-routine, ce qui peut provoquer des résultats non souhaités ou inatte ndus.

On Error Resume Next

L'instruction **On Error Resume Next** spécifie que dans l'éventualité d'une erreur d'exécution, le contrôle passe à l'instruction située immédiatement après celle où l'erreur s'est produite. À ce stade, l'exécution continue. **On Error Resume Next** vous permet de placer des routines de gestion d'erreur aux emplacements où les erreurs vont se produire, au lieu de transférer le contrôle à un autre emplacement dans la procédure.

Remarque

Si votre procédure en appelle une autre, l'instruction **On Error Resume Next** devient inactive durant l'exécution de la procé dure appelée. Par conséquent, vous devez placer une instruction **On Error Resume Next** dans chaque procédure appelée qu i le nécessite. Ceci est nécessaire, car le comportement de **Resume Next** ne s'applique qu'à la procédure contenant l'instructi on **On Error Resume Next**. Si une erreur non gérée se produit dans une procédure appelée, l'exception est retournée à la pr océdure appelante et l'exécution reprend à l'instruction située juste après l'appel. Dans ces cas-là, l'erreur n'est pas gérée.

L'instruction **Resume** peut également être utilisée de manière autonome, en dehors de l'instruction **On Error**. Lorsque **Resume** est utilisée de cette manière, Visual Basic retourne le contrôle à l'instruction à l'origine de l'erreur. **Resume** s'utilise généralement après qu'un gestionnaire d'erreur a corrigé l'erreur.

Visual Basic fournit également l'instruction **Resume Next**, qui dirige le contrôle vers la ligne située immédiatement après la ligne de code à l'origine de l'erreur. Vous pouvez utiliser **Resume Next** dans les cas où une erreur ne doit pas empêcher votre application de fonctionner. Vous pouvez également l'utiliser si une erreur ne peut pas modifier les résultats attendus de votre sous-routine.

Une autre variante de l'instruction **Resume** est **Resume Line**, qui est semblable à **On Error GoTo Line**. **Resume Line** passe le contrôle à une ligne que vous spécifiez dans l'argument *line*. Vous ne pouvez utiliser **Resume Line** que dans un gestionnaire d'erreur.

Remarque

Lorsque vous déboguez votre code, vous devez désactiver l'instruction **On Error Resume Next**.

On Error GoTo 0

L'instruction **On Error GoTo 0** désactive tout gestionnaire d'erreurs dans la procédure actuelle. Si vous n'ajoutez pas d'instruction **On Error GoTo 0**, le gestionnaire d'erreurs reste désactivé lorsque la procédure contenant le gestionnaire d'exceptions termine son exécution.

Remarque

L'instruction **On Error GoTo 0** ne spécifie pas la ligne 0 comme le début du code de gestion d'erreur, même si la procédure c ontient effectivement une ligne numérotée 0.

On Error GoTo -1

L'instruction **On Error GoTo -1** désactive tout gestionnaire d'exceptions dans la procédure actuelle. Si vous n'ajoutez pas d'instruction **On Error GoTo -1**, l'exception est automatiquement désactivée à la fin de l'exécution de la procédure en cours.

Remarque

L'instruction **On Error GoTo -1** ne spécifie pas la ligne -1 comme le début du code de gestion d'erreur, même si la procédure contient effectivement une ligne numérotée -1.

Exemple de gestionnaire d'exceptions non structuré

Dans le code suivant, le gestionnaire d'exceptions se nomme `DivideByZero` et gère une erreur spécifique (la tentative de division par zéro). Si une autre erreur se produit, Visual Basic génère une erreur d'exécution et arrête l'application.

```
Sub ErrorTest ()
' Declare variables.
   Dim x As Integer, y As Integer, z As Integer
   ' The exception handler is named "DivideByZero".
   On Error GoTo DivideByZero
   ' The main part of the code, which might cause an error.
   x = 2
   y = 0
   z = x \ y

   ' This line disables the exception handler.
   On Error GoTo 0
      Console.WriteLine(x & "/" & y & " = " & z)
```

```
    ' Exit the subroutine before the error-handling code.
    ' Failure to do so can create unexpected results.
    Exit Sub

    ' This is the exception handler, which deals with the error.
    DivideByZero:
    ' Include a friendly message to let the user know what is happening.
    Console.WriteLine("You have attempted to divide by zero!")

    ' Provide a solution to the error.
    y = 2

    ' The Resume statement returns to the point at which
    ' the error first occurred, so the application
    ' can continue to run.
    Resume

End Sub
```

Voir aussi

Tâches

Comment : extraire des informations à partir d'un objet Error

Référence

End, instruction
Err, objet (Visual Basic)
Exit, instruction (Visual Basic)
On Error, instruction (Visual Basic)
Resume, instruction

Concepts

Objet Err dans la gestion non structurée des exceptions
Introduction à la gestion des exceptions
Types d'erreurs
Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Objet Err dans la gestion non structurée des exceptions

Pour écrire un code de gestion d'erreur, vous devez déterminer l'erreur survenue. Dans la gestion des exceptions non structurée, vous pouvez utiliser pour cela l'Err, objet (Visual Basic).

L'objet **Err** contient des propriétés dont les valeurs sont déterminées par l'erreur qui vient de se produire. La propriété **Number** contient la cause de l'erreur et la propriété **Description** contient un message qui décrit l'erreur en détail. **Helpfile** et **HelpContext** vous permettent d'afficher un fichier d'aide connexe lorsqu'un utilisateur clique sur le bouton **Aide** ou appuie sur la touche F1. **LastDLLError** affiche la dernière DLL appelée et indique si l'appel a réussi ou échoué. **Source** spécifie une expression de chaîne qui représente l'objet ou l'application ayant généré l'erreur.

Votre gestionnaire d'erreur doit tester ou enregistrer les valeurs de propriétés appropriées avant qu'une autre erreur ne se produise. Votre code ne peut gérer qu'une seule erreur à la fois, par conséquent, vous devez traiter les erreurs les unes après les autres.

Remarque
L'objet **Err** peut être utilisé uniquement avec des exceptions interceptées avec l'instruction **On Error GoTo**

Erreurs spécifiques à l'application

Outre la gestion des erreurs fournie par Visual Basic, vous avez également la possibilité de gérer les erreurs spécifiques à votre application.

Si vous créez un objet qui accède à d'autres objets, vous devez écrire du code pour traiter les erreurs non gérées retournées par les autres objets. Si nécessaire, vous pouvez mapper les codes d'erreur de **Err.Number** vers l'une de vos erreurs spécifiques à l'application, puis retourner cette erreur à l'appelant de votre objet.

Voir aussi

Référence

On Error, instruction (Visual Basic)
Description, propriété (objet Err)
Number, propriété (objet Err)
HelpFile, propriété (objet Err)
HelpContext, propriété (objet Err)
LastDllError, propriété (objet Err)
Source, propriété (objet Err)

Concepts

Introduction à la gestion des exceptions
Types d'erreurs
Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic
Vue d'ensemble de la gestion non structurée des exceptions

Comment : extraire des informations à partir d'un objet Error

Lorsqu'une erreur d'exécution se produit, les propriétés de l'objet **Err** sont complétées par des informations qui identifient l'erreur de manière unique et par des informations qui peuvent être utilisées pour gérer l'erreur.

Les propriétés de l'objet **Err** sont réinitialisées à zéro ou à des chaînes de longueur nulle ("") après une instruction **On Error Resume Next** et après une instruction **Exit Sub** ou **Exit Function** dans une routine de gestion des erreurs. La méthode **Clear** peut être utilisée pour réinitialiser **Err** de manière explicite.

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide est orientée sur les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour récupérer des informations à partir d'un objet d'erreur

1. Vous pouvez filtrer des erreurs spécifiques. Cet exemple vérifie si l'erreur est une erreur FileNotFound et réagit si c'est le cas.

   VB

   ```
   If Err.Number = 53 Then
     MsgBox("File Not Found")
   End If
   ```

2. Vous pouvez examiner également des propriétés spécifiques de l'objet d'erreur, telles que **Description**, **Erl**, **HelpContext**, **Helpfile**, **LastDLLError**, **Number** et **Source**. Cet exemple affiche la description dans une boîte de message.

   VB

   ```
   MsgBox(Err.Description)
   ```

Voir aussi

Référence

Err, objet (Visual Basic)
Err, membres de l'objet
Description, propriété (objet Err)
Erl, propriété (objet Err)
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
Source, propriété (objet Err)

Concepts

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Déploiement de l'application

Le .NET Framework fournit plusieurs fonctionnalités de base qui facilitent le déploiement de diverses applications, notamment celui de ClickOnce.

Fonctionnalités de déploiement dans le .NET Framework

Les fonctionnalités de base suivantes du .NET Framework permettent de faciliter le déploiement :

- Applications sans impact
- Partage de code contrôlé
- Versioning côte à côte
- Mises à jour à la volée
- Code de confiance partielle

Pour plus d'informations, consultez Fonctionnalités de déploiement du .NET Framework.

Notions de base du déploiement

Le déploiement est le processus selon lequel une application ou un composant fini est distribué en vue de son installation sur d'autres ordinateurs. Dans Visual Studio, le déploiement peut être effectué à l'aide de la technologie Microsoft Windows Installer ou de la technologie ClickOnce. Pour plus d'informations, consultez Choix d'une stratégie de déploiement.

Une application peut être empaquetée comme un assembly unique, une collection d'assemblys, des fichiers CAB (cabinet), un package Microsoft Windows Installer 2.0 ou comme un package de programme d'installation d'un autre format. Pour plus d'informations, consultez Scénarios de déploiement pour les applications .NET Framework.

Un assembly est une collection de types et de ressources qui sont générés pour fonctionner ensemble et former une unité logique de fonctionnalités. Les assemblys constituent l'unité fondamentale de déploiement, le contrôle des versions, la réutilisation, la portée d'activation et les autorisations de sécurité. Ils fournissent le Common Language Runtime ainsi que les informations nécessaires à la reconnaissance des implémentations des types.

Ce tableau affiche les tâches de déploiement courantes.

Pour	**Reportez-vous à**
Créer un assembly	Création d'assemblys
Signer un assembly avec un nom fort	Comment : signer un assembly avec un nom fort
Signer un assembly, mais différer à plus tard l'assignation à une clé privée	Comment : temporiser la signature d'un assembly (Visual Studio)
Créer une paire de clés publique/privée à utiliser pour la création d'un nom fort	Comment : créer une paire de clés publique/privée
Consulter les informations de liaison dans le fichier journal	Assembly Binding Log Viewer (Fuslogvw.exe)
Déployer une application Windows Forms	Comment : développer un contrôle Windows Forms simple
Déployer une application pour accéder à COM	Déploiement d'une application pour accéder à COM
Déployer un service Web XML	Déploiement de services Web XML
Déployer une application .NET Framework	Déploiement d'applications .NET Framework

Versioning

Le versioning, qui enregistre la version spécifique d'un assembly et les versions des assemblys dépendants du manifeste de

l'assembly, ne s'effectue que sur les assemblys portant des noms forts. Les assemblys à nom fort ont des identificateurs qui se composent de l'identité de l'assembly, à laquelle s'ajoute une clé publique et une signature numérique.

Par défaut, le CLR essaie de rechercher et de se lier à la version exacte de l'assembly avec laquelle l'application a été générée, bien que ce comportement puisse être substitué par les paramètres de fichier de configuration.

Pour plus d'informations, consultez Assemblys dans le Common Language Runtime, Versioning des assemblys et Assemblys avec nom fort.

Déploiement ClickOnce

Le déploiement ClickOnce vous permet de publier des applications Windows sur un serveur Web ou un partage de fichier réseau pour simplifier l'installation. Lorsque vous souhaitez publier votre application Windows sur un serveur réseau, Visual Studio génère automatiquement les fichiers manifeste XML nécessaires pour ClickOnce et publie l'application sur le serveur spécifié.

Les applications ClickOnce sont indépendantes et s'installent sur une base individuelle, ce qui signifie qu'aucun droit d'administration n'est requis. Les applications déployées à l'aide de la technologie ClickOnce sont exécutées en sandbox, avec un jeu limité d'autorisations basé sur une zone de sécurité. Pour plus d'informations, consultez Déploiement et sécurité ClickOnce.

Les applications ClickOnce peuvent être mises à jour automatiquement ; vous pouvez contrôler quand et comment les mises à jour sont déployées.

Ce tableau affiche les tâches associées au déploiement ClickOnce.

Pour	Reportez-vous à
Choisir quelle stratégie de déploiement ClickOnce utiliser avec votre application : installation à partir du Web ou d'un partage réseau, installation à partir d'un CD ou démarrage à partir du Web ou d'un partage réseau	Choix d'une stratégie de déploiement ClickOnce
Choisir quelle stratégie de mise à jour ClickOnce utiliser avec votre application	Choix d'une stratégie de mise à jour ClickOnce
Déployer une application ClickOnce	Comment : publier une application ClickOnce
Choisir s'il faut utiliser un répertoire de données ClickOnce, un stockage isolé ou d'autres fichiers locaux lors de la lecture et de l'écriture des données	Accès aux données locales et distantes dans les applications ClickOnce
Connaître les composants requis pour déployer avec succès une application	Déploiement des composants requis (Visual Studio)
Spécifier un autre emplacement pour les mises à jour de déploiement	Comment : spécifier un autre emplacement pour les mises à jour du déploiement
Élever le niveau d'autorisation d'une application ClickOnce sur un ordinateur client	Vue d'ensemble du déploiement d'applications approuvées

Les difficultés rencontrées lors du déploiement d'applications à l'aide de ClickOnce peuvent être causées par les problèmes de compression HTTP, d'analyse du manifeste, de configuration du serveur et de versioning. Pour plus d'informations, consultez Dépannage du déploiement ClickOnce.

Déploiement de Windows Installer

La technologie Microsoft Windows Installer crée des programmes d'installation qui vous permettent de contrôler entièrement la procédure d'installation.

Ce tableau décrit les tâches associées au déploiement Windows Installer.

Pour	Reportez-vous à
Distribuer des fichiers ou des applications sur un CD-ROM ou un autre média	Déploiement via des supports distribuables
Créer un projet d'installation ou ajouter un projet de déploiement existant à une solution	Comment : créer ou ajouter un projet d'installation

Créer un fichier .cab pour le téléchargement Web ou en ajouter un qui existe à une solution	Comment : créer ou ajouter un projet CAB
Créer un module de fusion pour un composant	Comment : ajouter des modules de fusion à un projet de déploiement

Déployer une application Windows sur un autre ordinateur	Procédure pas à pas : déploiement d'une application Windows
Déployer un site Web sur un serveur Web	Déploiement de sites Web ASP.NET (Visual Studio)
Créer une action personnalisée pour envoyer un utilisateur sur un site Web après l'installation	Procédure pas à pas : création d'une action personnalisée

Passer des données à une propriété dynamique lors de l'installation en utilisant une action personnalisée	Procédure pas à pas : utilisation d'une action personnalisée pour afficher un message au cours de l'installation
Utiliser une action personnalisée pour précompiler un assembly au cours de l'installation	Procédure pas à pas : utilisation d'une action personnalisée pour précompiler un assembly au cours de l'installation

Utiliser une action personnalisée pour créer une base de données au cours de l'installation	Procédure pas à pas : utilisation d'une action personnalisée pour créer une base de données au moment de l'installation
Créer une application Web qui peut être redirigée pour cibler un service Web XML différent	Procédure pas à pas : redirection d'une application afin de cibler un service Web XML différent au moment de l'installation

Déployer des fichiers et des applications sur un serveur Web	Déploiement d'un projet d'installation Web

Voir aussi

Concepts

Autres solutions de déploiement

Autres ressources

Procédures pas à pas relatives au déploiement de Windows Installer

Création et utilisation de composants en Visual Basic

Un *composant* est une classe qui implémente l'interface System.ComponentModel.IComponent ou qui dérive directement ou indirectement d'une classe qui implémente **IComponent**. Un composant du .NET Framework est un objet réutilisable qui peut interagir avec d'autres objets, permet de contrôler les ressources externes et offre une prise en charge au moment du design.

Une fonctionnalité importante des composants est leur concevabilité, ce qui signifie qu'une classe qui est un composant peut être utilisée dans un environnement de développement intégré Visual Studio. Un composant peut être ajouté à la Boîte à outils, faire l'objet d'un glisser-déplacer sur un formulaire et être manipulé sur une aire de conception. Notez que la prise en charge de base au moment du design pour les composants est construite dans le .NET Framework ; un développeur de composant n'a aucun travail supplémentaire à effectuer pour profiter de la fonctionnalité de base au moment du design.

Un *contrôle* est similaire à un composant, dans la mesure où tous les deux sont concevables. Toutefois, un contrôle fournit une interface utilisateur, ce qui n'est pas le cas d'un composant. Un contrôle doit dériver de l'une des classes de contrôle de base : Control ou Control.

Moment de création d'un composant

Si votre classe doit être utilisée sur une aire de conception (telle que le concepteur Windows Forms ou Web Forms), et qu'elle n'a pas d'interface, elle doit être un composant et implémenter **IComponent** ou dériver d'une classe qui implémente directement ou indirectement **IComponent**.

Les classes Component et MarshalByValueComponent constituent des implémentations de base de l'interface **IComponent**. La différence principale entre ces classes est que la classe **Component** est marshalée par référence, tandis que la classe **IComponent** est marshalée par valeur. La liste suivante fournit des indications générales pour les implémenteurs.

- Si votre composant doit être marshalé par référence, dérivez de **Component**.
- Si votre composant doit être marshalé par valeur, dérivez de **MarshalByValueComponent**.
- Si votre composant ne peut pas dériver de l'une des implémentations de base en raison d'un héritage unique, implémentez **IComponent**.

Pour plus d'informations sur la prise en charge au moment du design, consultez
Attributs en mode design pour les composants et Extension de la prise en charge au moment du design.

Classes de composant

L'espace de noms System.ComponentModel fournit des classes qui sont utilisées pour implémenter le comportement au moment de l'exécution et au moment du design des composants et des contrôles. Cet espace de noms inclut les classes de base et les interfaces pour l'implémentation des attributs et des convertisseurs de type, pour la liaison à des sources de données et pour la licence des composants.

Les principales classes de composant sont les suivantes :

- **Component**. Implémentation de base pour l'interface **IComponent**. Cette classe permet le partage d'objets entre applications.
- **MarshalByValueComponent**. Implémentation de base pour l'interface **IComponent**.
- Container. Implémentation de base pour l'interface IContainer. Cette classe encapsule aucun ou plusieurs composants.

Les classes utilisées pour la licence des composants sont entre autres les suivantes :

- License. Classe de base abstraite pour toutes les licences. Une licence est accordée à une instance spécifique d'un composant.
- LicenseManager. Fournit des propriétés et des méthodes pour ajouter une licence à un composant et pour gérer un LicenseProvider.
- **LicenseProvider**. Classe de base abstraite pour l'implémentation d'un fournisseur de licences.
- LicenseProviderAttribute. Spécifie la classe **LicenseProvider** à utiliser avec une classe.

Classes fréquemment utilisées pour la description et la persistance des composants.

- TypeDescriptor. Fournit des informations sur les caractéristiques d'un composant, telles que ses attributs, ses propriétés

et ses événements.

- EventDescriptor. Fournit des informations sur un événement.
- PropertyDescriptor. Fournit des informations sur une propriété.

Rubriques connexes

Comparaison entre classe, composant et contrôle

Définit le *composant* et le *contrôle* et décrit les différences entre eux et les classes.

Création de composants

Documentation pour la mise en route des composants.

Procédures pas à pas de la création de composants

Propose des liens vers des rubriques fournissant des instructions étape par étape sur la programmation de composants.

Classes de composant

Explique comment une classe devient un composant, comment exposer les fonctionnalités des composants et comment contrôler l'accès aux composants ainsi que la création des instances de composants.

Dépannage de la création de contrôles et de composants

Explique comment résoudre certains problèmes courants.

Voir aussi

Tâches

Comment : accéder à la prise en charge au moment du design dans les Windows Forms
Comment : étendre l'apparence et le comportement des contrôles en mode design
Comment : effectuer une initialisation personnalisée pour les contrôles en mode design

Fonctionnalités plus poussées de Visual Basic

Même si la fonctionnalité **My** offre une large gamme de fonctions, vous pouvez avoir envie d'aller au-delà de **My** et explorer les autres fonctionnalités que propose le .NET Framework. Ces pages fournissent des vues d'ensemble et des pointeurs vers la documentation appropriée des zones .NET Framework, y compris des formulaires, des opérations de réseau et des services Web.

Dans cette section

Authentification et autorisation dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework pour l'authentification et l'autorisation des utilisateurs.

Autorisations et sécurité d'accès au code dans le .NET Framework avec Visual Basic

Décrit comment la sécurité et les autorisations d'accès au code fonctionnent dans le .NET Framework.

Collections dans le .NET Framework avec Visual Basic

Décrit les classes spécialisées pour le stockage et la récupération de données par le .NET Framework.

E/S de fichier dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework pour l'E/S de fichier.

Graphiques dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework lors de l'utilisation des graphiques.

Enregistrement et traçage dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework lors de la création et de la manipulation des journaux ou du traçage.

Personnalisation des projets et extension de My avec Visual Basic

Décrit comment spécifier vos propres extensions au modèle d'application et personnaliser les modèles de projet pour fournir des objets **My** supplémentaires.

Opérations de réseau dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework lors de l'exécution des opérations de réseau.

Opérations relatives au port dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework lors de l'exécution des opérations de port.

Sérialisation dans le .NET Framework avec Visual Basic

Décrit comment utiliser le .NET Framework pour la sérialisation.

Développement pour les Smart Devices avec Visual Basic

Fournit une vue d'ensemble des concepts de base de la programmation des périphériques intelligents.

Multithreading avancé avec Visual Basic

Fournit une vue d'ensemble du modèle de thread.

Concepts d'applications Windows Forms en Visual Basic

Fournit une vue d'ensemble des applications Windows.

Ajout de rapports affichables aux applications Visual Studio

Discute des options disponibles pour écrire des rapports.

Authentification et autorisation dans le .NET Framework avec Visual Basic

Le .NET Framework fournit une infrastructure extensible pour l'autorisation et l'authentification des utilisateurs. Les rubriques de cette section fournissent une vue d'ensemble de ces concepts et répertorient les classes couramment utilisées

Dans cette section

Concepts de base des opérations de l'utilisateur du .NET Framework

Décrit comment utiliser l'autorisation d'applications dans le .NET Framework et comment utiliser la sécurité basée sur les rôles.

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Montre comment implémenter une authentification et une autorisation personnalisées et comment substituer l'identité par défaut du thread de l'application.

Classes utilisées dans les opérations d'utilisateur du .NET Framework

Répertorie les classes utilisées dans le .NET Framework qui forment la base du système d'authentification utilisateur.

Voir aussi

Référence

My.User, objet

System.Security.Principal

Concepts

Accès aux données de l'utilisateur

Autres ressources

Fonctionnalités plus poussées de Visual Basic

Concepts de base des opérations de l'utilisateur du .NET Framework

Le .NET Framework fournit une implémentation de sécurité basée sur les rôles dans l'espace de noms System.Security.Principal que vous pouvez utiliser pour autoriser et authentifier les utilisateurs de votre application. Cette rubrique décrit comment utiliser l'autorisation d'application dans le .NET Framework et créer des objets IIdentity et IPrincipal pour représenter les utilisateurs.

Un **IIdentity** encapsule un utilisateur authentifié. Un **IPrincipal** est une combinaison de l'identité de l'utilisateur et des rôles qu'il a. Vous pouvez utiliser les classes Identity et Principal prédéfinies dans l'espace de noms **System.Security.Principal** ou ajouter une authentification personnalisée en créant des classes qui implémentent les interfaces.

Pour utiliser ces interfaces, vous devez qualifier pleinement les noms d'interface ou importer les espaces de noms appropriés en incluant les instructions **Imports** au début du fichier de code source affecté. Pour plus d'informations, consultez Imports, instruction.

Utilisation de la sécurité basée sur les rôles dans les applications .NET Framework

Chaque thread d'application peut avoir un objet Principal associé (accessible via la propriété CurrentPrincipal) qui représente le contexte de sécurité de l'utilisateur pour lequel le code s'exécute. Un objet Principal encapsule l'objet (ou identité) du contexte de sécurité de l'utilisateur. L'utilisation de deux objets permet une séparation de l'authentification (dans l'objet Identity) et de l'autorisation (dans l'entité de sécurité).

Un objet Identity doit implémenter l'interface **IIdentity**. Un objet Identity représente un utilisateur particulier et expose les propriétés suivantes requises par l'interface **IIdentity** : Name, IsAuthenticated et AuthenticationType. Les objets Identity ont en général des membres privés supplémentaires qui effectuent l'authentification utilisateur.

Un objet Principal doit implémenter l'interface **IPrincipal**. Un objet Principal encapsule le contexte de sécurité de l'utilisateur en exposant les membres suivants requis par l'interface **IPrincipal** : la méthode IsInRole qui exécute l'autorisation et la propriété Identity qui fournit l'accès à l'objet Identity de l'utilisateur.

Utilisation des identités

Le .NET Framework fournit quatre classes qui implémentent l'interface **IIdentity** :

- WindowsIdentity
- GenericIdentity
- PassportIdentity
- FormsIdentity

Chaque classe vous permet d'utiliser différents types d'identités utilisateur. Pour accéder à l'objet **WindowsIdentity** actuel d'une application qui utilise l'authentification Windows, utilisez la méthode GetCurrent statique de la classe **WindowsIdentity**. Vous pouvez également définir l'entité de sécurité du thread actuel en appelant la méthode My.User.InitializeWithWindowsUser.

Vous pouvez également créer des classes Identity personnalisées en implémentant l'interface **IIdentity** dans votre classe personnalisée. Pour plus d'informations sur la création d'identités personnalisées, consultez Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées.

Utilisation des entités de sécurité

Le .NET Framework fournit l'interface **IPrincipal** pour lier les rôles et les identités d'utilisateur. Si votre application effectue l'autorisation, elle doit utiliser un objet qui implémente **IPrincipal**. Par exemple, les classes **WindowsIdentity** et **GenericIdentity** fournissent des implémentations intégrées de **IPrincipal**. Vous pouvez également créer vos propres classes Principal personnalisées selon **IPrincipal**.

Vous pouvez lier le thread actuel à un objet **IPrincipal** en assignant l'objet à la propriété **CurrentPrincipal** du thread ou à la propriété My.User.CurrentPrincipal. Vous pouvez ensuite effectuer une vérification d'autorisation en testant si l'utilisateur est membre d'un rôle particulier. Pour ce faire, utilisez la méthode **IsInRole** de l'entité de sécurité.

Les applications ASP.NET gèrent les objets **IPrincipal** différemment par rapport aux autres applications .NET Framework. ASP.NET crée l'apparence d'une session sur le protocole HTTP sans état. Dans le cadre de cette session, l'objet **IPrincipal** qui

représente l'utilisateur est disponible dans la propriété User de l'objet HttpContext pour tout le code qui exécute la demande de l'utilisateur. Le Common Language Runtime met automatiquement à jour **CurrentPrincipal** avec la valeur **User** après l'événement OnAuthenticate du fichier Global.asax. Les applications ASP.NET utilisent souvent la propriété **User** pour effectuer des vérifications d'autorisation.

Remarque
La modification manuelle de **User** met automatiquement à jour la propriété **CurrentPrincipal** pour tous les threads qui s'exécutent dans le même contexte HTTP. Toutefois, la modification de **CurrentPrincipal** n'affecte pas la propriété **User**. Elle n'affecte que le thread choisi pour le reste de la demande.

Dans les applications ASP.NET, la propriété My.User.CurrentPrincipal met à jour la propriété **User**.

Pour plus d'informations sur la création de types **IPrincipal** personnalisés, consultez Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées.

Octroi d'autorisations pour utiliser les objets IIdentity et IPrincipal

Vous devez agir avec précaution lorsque vous accordez des autorisations pour utiliser les objets **IIdentity**, parce que ces objets rendent disponibles des informations sensibles relatives à l'utilisateur. Vous devez protéger l'objet **IPrincipal** actuel de l'application contre les modifications parce que la fonction d'autorisation de l'application repose sur son entité de sécurité actuelle.

Le .NET Framework fournit cette protection en exigeant que ces opérations aient une autorisation de sécurité d'accès de code. Accordez l'autorisation System.Security.Permissions.SecurityPermissionAttribute.ControlPrincipal aux applications qui doivent manipuler ces objets en utilisant Outil Code Access Security Policy Tool (Caspol.exe).

Par défaut, toutes les applications installées localement ont cette autorisation parce qu'elles s'exécutent sous le jeu d'autorisations FullTrust.

L'exécution des méthodes suivantes requiert l'autorisation **ControlPrincipal** :

- System.AppDomain.SetPrincipalPolicy(System.Security.Principal.PrincipalPolicy)
- **System.Security.Principal.WindowsIdentity.GetCurrent**
- System.Security.Principal.WindowsIdentity.Impersonate
- **System.Threading.Thread.CurrentPrincipal**

Voir aussi

Tâches

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet

Autres ressources

Authentification et autorisation dans le .NET Framework avec Visual Basic

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Cette procédure pas à pas montre comment implémenter une authentification et une autorisation personnalisées à l'aide de classes qui dérivent de IIdentity et de IPrincipal. Elle montre également comment substituer l'identité par défaut du thread de l'application, l'identité Windows, en affectant à **My.User.CurrentPrincipal** une instance de la classe qui dérive de **IPrincipal**. Les nouvelles informations utilisateur sont immédiatement disponibles via l'objet **My.User** qui retourne des informations sur l'identité de l'utilisateur en cours du thread.

Les applications destinées aux entreprises accordent souvent l'accès aux données et aux ressources en fonction des informations d'identification fournies par l'utilisateur. Pour cela, elles vérifient en général le rôle de l'utilisateur afin de lui donner accès aux ressources en rapport avec son rôle. Le Common Language Runtime fournit une prise en charge des autorisations basées sur les rôles pour des comptes Windows ou des identités personnalisées. Pour plus d'informations, consultez Sécurité basée sur les rôles.

Mise en route

Pour commencer, installez un projet avec un formulaire principal et un formulaire de connexion et configurez-le pour utiliser une authentification personnalisée.

Pour créer l'exemple d'application

1. Créez un projet d'application Windows Visual Basic. Pour plus d'informations, consultez Comment : créer un projet d'application Windows.

   Le nom par défaut du formulaire principal est `Form1`.

2. Dans le menu **Projet**, cliquez sur **Ajouter un nouvel élément**.

3. Sélectionnez le modèle **Formulaire de connexion** et cliquez sur **Ajouter**.

   Le nom par défaut du formulaire de connexion est `LoginForm1`.

4. Dans le menu **Projet**, cliquez sur **Ajouter un nouvel élément**.

5. Sélectionnez le modèle **Classe**, remplacez le nom par `SampleIIdentity`, puis cliquez sur **Ajouter**.

6. Dans le menu **Projet**, cliquez sur **Ajouter un nouvel élément**.

7. Sélectionnez le modèle **Classe**, remplacez le nom par `SampleIPrincipal`, puis cliquez sur **Ajouter**.

8. Dans le menu **Projet**, cliquez sur **Propriétés <NomApplication>**.

9. Dans le Concepteur de projets, cliquez sur l'onglet **Application**.

10. Remplacez la liste déroulante **Mode d'authentification** par **Défini au niveau de l'application**.

Pour configurer le formulaire principal

1. Basculez vers `Form1` dans le Concepteur de formulaires.

2. Ajoutez un **Bouton** à `Form1` dans la **Boîte à outils**.

   Le nom par défaut du bouton est `Button1`.

3. Remplacez le texte du bouton par **Authentifier**.

4. Dans la **Boîte à outils**, ajoutez une **Étiquette** à `Form1`.

   Le nom par défaut de l'étiquette est `Label1`.

5. Remplacez le texte de l'étiquette par une chaîne vide.

6. Dans la **Boîte à outils**, ajoutez une **Étiquette** à Form1.

   Le nom par défaut de l'étiquette est `Label2`.

7. Remplacez le texte de l'étiquette par une chaîne vide.
8. Double-cliquez sur **Button1** pour créer le gestionnaire d'événements pour l'événement Click, puis ouvrez l'éditeur de code.
9. Ajoutez le code suivant à la méthode `Button1_Click`.

   VB

```
My.Forms.LoginForm1.ShowDialog()
' Check if the user was authenticated.
If My.User.IsAuthenticated Then
    Me.Label1.Text = "Authenticated " & My.User.Name
Else
    Me.Label1.Text = "User not authenticated"
End If

If My.User.IsInRole(ApplicationServices.BuiltInRole.Administrator) Then
    Me.Label2.Text = "User is an Administrator"
Else
    Me.Label2.Text = "User is not an Administrator"
End If
```

Vous pouvez lancer l'application, mais étant donné qu'elle ne contient pas de code d'authentification, elle n'authentifiera pas d'utilisateur. L'ajout d'un code d'authentification est abordé dans la section suivante.

Création d'une identité

Le .NET Framework utilise les interfaces **IIdentity** et **IPrincipal** comme la base de l'authentification et de l'autorisation. Votre application peut utiliser une authentification utilisateur personnalisée en implémentant ces interfaces, comme le montrent ces procédures.

Pour créer une classe qui implémente IIdentity

1. Sélectionnez le fichier SampleIIdentity.vb dans l'**Explorateur de solutions**.

   Cette classe encapsule l'identité d'un utilisateur.

2. Dans la ligne `Public Class SampleIIdentity` suivante, ajoutez le code suivant pour hériter de **IIdentity**.

   VB

```
Implements System.Security.Principal.IIdentity
```

   Une fois que vous avez ajouté le code et appuyé sur ENTRÉE, l'éditeur de code crée les propriétés stub que vous devez implémenter.

3. Ajoutez des champs privés pour stocker le nom d'utilisateur et une valeur qui indique si l'utilisateur est authentifié.

   VB

```
Private nameValue As String
Private authenticatedValue As Boolean
Private roleValue As ApplicationServices.BuiltInRole
```

4. Entrez le code suivant dans la propriété AuthenticationType.

   La propriété **AuthenticationType** doit retourner une chaîne qui indique le mécanisme d'authentification actuel.

   Cet exemple utilise l'authentification spécifiée explicitement, donc la chaîne est "Authentification personnalisée". Si les données d'authentification de l'utilisateur ont été stockées dans une base de données SQL Server, la valeur peut être

"SqlDatabase".

VB

```
Return "Custom Authentication"
```

5. Entrez le code suivant dans la propriété IsAuthenticated.

   VB

   ```
   Return authenticatedValue
   ```

   La propriété **IsAuthenticated** doit retourner une valeur qui indique si l'utilisateur a été authentifié.

6. La propriété Name doit retourner le nom de l'utilisateur associé à cette identité.

   Entrez le code suivant dans la propriété **Name**.

   VB

   ```
   Return nameValue
   ```

7. Créez une propriété qui retourne le rôle de l'utilisateur.

   VB

   ```
   Public ReadOnly Property Role() As ApplicationServices.BuiltInRole
       Get
           Return roleValue
       End Get
   End Property
   ```

8. Créez une méthode **Sub New** qui initialise la classe en authentifiant l'utilisateur, puis en définissant son nom et son rôle, sur la base d'un nom et d'un mot de passe.

   Cette méthode appelle une méthode nommée `IsValidNameAndPassword` pour déterminer si une association nom d'utilisateur/mot de passe est valide.

   VB

   ```
   Public Sub New(ByVal name As String, ByVal password As String)
       ' The name is not case sensitive, but the password is.
       If IsValidNameAndPassword(name, password) Then
           nameValue = name
           authenticatedValue = True
           roleValue = ApplicationServices.BuiltInRole.Administrator
       Else
           nameValue = ""
           authenticatedValue = False
           roleValue = ApplicationServices.BuiltInRole.Guest
       End If
   End Sub
   ```

9. Créez une méthode nommée `IsValidNameAndPassword` qui détermine si une association nom d'utilisateur/mot de passe est valide.

   Remarque de sécurité

L'algorithme d'authentification doit traiter les mots de passe de manière sécurisée. Par exemple, le mot de passe ne doi t pas être stocké dans un champ de classe.

Vous ne devez pas stocker les mots de passe utilisateur dans votre système, parce que si ces informations sont perdues, il n'y a plus de sécurité. Vous pouvez stocker le *hachage* du mot de passe de chaque utilisateur. Une fonction de hachage brouille les données de sorte que l'entrée ne puisse pas être déduite de la sortie. Un mot de passe ne peut pas être déterminé directement à partir du hachage d'un autre.

Toutefois, un utilisateur malveillant pourrait prendre le temps de générer un dictionnaire des hachages de tous les mots de passe possibles, puis rechercher un hachage donné d'un mot de passe. Pour se prémunir contre ce type d'attaque, vous devez ajouter un *salt* au mot de passe avant de le hacher pour générer un hachage salted. Le salt est constitué de données supplémentaires uniques à chaque mot de passe, empêchant ainsi le précalcul d'un dictionnaire de hachage.

Pour protéger les mots de passe contre les utilisateurs malveillants, vous ne devez stocker que des *hachages salted* des mots de passe, de préférence sur un ordinateur sécurisé. Il est très difficile pour un utilisateur malveillant de récupérer un mot de passe d'un hachage salted. Cet exemple utilise les méthodes `GetHashedPassword` et `GetSalt` pour charger le mot de passe haché et le salt d'un utilisateur.

VB

```
Private Function IsValidNameAndPassword( _
    ByVal username As String, _
    ByVal password As String) _
    As Boolean

    ' Look up the stored hashed password and salt for the username.
    Dim storedHashedPW As String = GetHashedPassword(username)
    Dim salt As String = GetSalt(username)

    'Create the salted hash.
    Dim rawSalted As String = salt & Trim(password)
    Dim saltedPwBytes() As Byte = _
        System.Text.Encoding.Unicode.GetBytes(rawSalted)
    Dim sha1 As New _
        System.Security.Cryptography.SHA1CryptoServiceProvider
    Dim hashedPwBytes() As Byte = sha1.ComputeHash(saltedPwBytes)
    Dim hashedPw As String = Convert.ToBase64String(hashedPwBytes)

    ' Compare the hashed password with the stored password.
    Return hashedPw = storedHashedPW
End Function
```

10. Créez les fonctions appelées `GetHashedPassword` et `GetSalt` qui retournent le mot de passe haché et le salt pour l'utilisateur spécifié.

Remarque de sécurité

Vous devez éviter le codage en dur des mots de passe hachés et des salts dans vos applications clientes pour deux rais ons. D'abord, des utilisateurs malveillants peuvent y accéder et trouver une collision de hachage. Ensuite, vous ne pouv ez pas modifier ni révoquer le mot de passe d'un utilisateur. L'application doit obtenir le mot de passe haché et le salt p our un utilisateur donné à partir d'une source sécurisée gérée par un administrateur.

Même si, pour des raisons de simplicité, cet exemple a un mot de passe haché et un salt codés en dur, vous devez utiliser une approche plus sécurisée dans le code de production. Par exemple, vous pouvez stocker les informations utilisateur dans une base de données SQL Server et y accéder à l'aide de procédures stockées. Pour plus d'informations, consultez Comment : établir une connexion à des données d'une base de données.

Remarque

Le mot de passe correspondant à ce mot de passe haché codé en dur est fourni dans la section « Test de l'application ».

VB

```
    Private Function GetHashedPassword(ByVal username As String) As String
        ' Code that gets the user's hashed password goes here.
        ' This example uses a hard-coded hashed passcode.
        ' In general, the hashed passcode should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "ZFFzgfsGjgtmExzWBRmZI5S4w6o="
        Else
            Return ""
        End If
    End Function

    Private Function GetSalt(ByVal username As String) As String
        ' Code that gets the user's salt goes here.
        ' This example uses a hard-coded salt.
        ' In general, the salt should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "Should be a different random value for each user"
        Else
            Return ""
        End If
    End Function
```

Le fichier SampleIIdentity.vb doit désormais contenir le code suivant :

VB

```
Public Class SampleIIdentity
    Implements System.Security.Principal.IIdentity

    Private nameValue As String
    Private authenticatedValue As Boolean
    Private roleValue As ApplicationServices.BuiltInRole

    Public ReadOnly Property AuthenticationType() As String Implements System.Security.Prin
cipal.IIdentity.AuthenticationType
        Get
            Return "Custom Authentication"
        End Get
    End Property

    Public ReadOnly Property IsAuthenticated() As Boolean Implements System.Security.Princi
pal.IIdentity.IsAuthenticated
        Get
            Return authenticatedValue
        End Get
    End Property

    Public ReadOnly Property Name() As String Implements System.Security.Principal.IIdentit
y.Name
        Get
            Return nameValue
        End Get
    End Property

    Public ReadOnly Property Role() As ApplicationServices.BuiltInRole
        Get
```

```
            Return roleValue
        End Get
    End Property

    Public Sub New(ByVal name As String, ByVal password As String)
        ' The name is not case sensitive, but the password is.
        If IsValidNameAndPassword(name, password) Then
            nameValue = name
            authenticatedValue = True
            roleValue = ApplicationServices.BuiltInRole.Administrator
        Else
            nameValue = ""
            authenticatedValue = False
            roleValue = ApplicationServices.BuiltInRole.Guest
        End If
    End Sub

    Private Function IsValidNameAndPassword( _
        ByVal username As String, _
        ByVal password As String) _
        As Boolean

        ' Look up the stored hashed password and salt for the username.
        Dim storedHashedPW As String = GetHashedPassword(username)
        Dim salt As String = GetSalt(username)

        'Create the salted hash.
        Dim rawSalted As String = salt & Trim(password)
        Dim saltedPwBytes() As Byte = _
            System.Text.Encoding.Unicode.GetBytes(rawSalted)
        Dim sha1 As New _
            System.Security.Cryptography.SHA1CryptoServiceProvider
        Dim hashedPwBytes() As Byte = sha1.ComputeHash(saltedPwBytes)
        Dim hashedPw As String = Convert.ToBase64String(hashedPwBytes)

        ' Compare the hashed password with the stored password.
        Return hashedPw = storedHashedPW
    End Function

    Private Function GetHashedPassword(ByVal username As String) As String
        ' Code that gets the user's hashed password goes here.
        ' This example uses a hard-coded hashed passcode.
        ' In general, the hashed passcode should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "ZFFzgfsGjgtmExzWBRmZI5S4w6o="
        Else
            Return ""
        End If
    End Function

    Private Function GetSalt(ByVal username As String) As String
        ' Code that gets the user's salt goes here.
        ' This example uses a hard-coded salt.
        ' In general, the salt should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "Should be a different random value for each user"
        Else
            Return ""
        End If
    End Function

End Class
```

Création d'une entité de sécurité

Ensuite, vous devez implémenter une classe qui dérive de **IPrincipal** et la configurer pour qu'elle retourne des instances de la classe `SampleIIdentity`.

Pour créer une classe qui implémente IPrincipal

1. Sélectionnez le fichier SampleIPrincipal.vb dans l'**Explorateur de solutions**.

   Cette classe encapsule l'identité d'un utilisateur. Vous pouvez utiliser l'objet **My.User** pour joindre cette entité de sécurité au thread actuel et accéder à l'identité de l'utilisateur.

2. Dans la ligne `Public Class SampleIPrincipal` suivante, ajoutez le code suivant pour hériter de **IPrincipal**.

   VB

   ```
   Implements System.Security.Principal.IPrincipal
   ```

   Une fois que vous avez ajouté le code et appuyé sur ENTRÉE, créez une propriété et une méthode stub que vous devez implémenter.

3. Ajoutez un champ privé pour stocker l'identité associée à cette entité de sécurité.

   VB

   ```
   Private identityValue As SampleIIdentity
   ```

4. Entrez le code suivant dans la propriété Identity.

   VB

   ```
   Return identityValue
   ```

   La propriété **Identity** doit retourner l'identité de l'utilisateur de l'entité de sécurité actuelle.

5. Entrez le code suivant dans la méthode IsInRole.

   La méthode **IsInRole** détermine si l'entité de sécurité actuelle appartient au rôle spécifié.

   VB

   ```
   Return role = identityValue.Role.ToString
   ```

6. Créez une méthode **Sub New** qui initialise la classe à l'aide d'une nouvelle instance de `SampleIIdentity` avec un nom d'utilisateur et un mot de passe.

   VB

   ```
   Public Sub New(ByVal name As String, ByVal password As String)
       identityValue = New SampleIIdentity(name, password)
   End Sub
   ```

   Ce code définit l'identité de l'utilisateur pour la classe `SampleIPrincipal`.

Le fichier SampleIPrincipal.vb doit désormais contenir le code suivant :

VB

```
Public Class SampleIPrincipal
    Implements System.Security.Principal.IPrincipal

    Private identityValue As SampleIIdentity
```

```
    Public ReadOnly Property Identity() As System.Security.Principal.IIdentity Implements S
ystem.Security.Principal.IPrincipal.Identity
        Get
            Return identityValue
        End Get
    End Property

    Public Function IsInRole(ByVal role As String) As Boolean Implements System.Security.Pr
incipal.IPrincipal.IsInRole
        Return role = identityValue.Role.ToString
    End Function

    Public Sub New(ByVal name As String, ByVal password As String)
        identityValue = New SampleIIdentity(name, password)
    End Sub

End Class
```

Connexion du formulaire de connexion

L'application peut utiliser le formulaire de connexion pour collecter un nom d'utilisateur et un mot de passe. Elle peut utiliser ces informations pour initialiser une instance de la classe `SampleIPrincipal` et utiliser l'objet **My.User** pour affecter cette instance à l'identité du thread actuel.

Pour configurer le formulaire de connexion

1. Sélectionnez `LoginForm1` dans le concepteur.
2. Double-cliquez sur le bouton **OK** pour ouvrir l'éditeur de code pour l'événement **Click**.
3. Remplacez le code dans la méthode `OK_Click` par le code suivant :

   VB

```
Dim samplePrincipal As New SampleIPrincipal( _
    Me.UsernameTextBox.Text, Me.PasswordTextBox.Text)
Me.PasswordTextBox.Text = ""
If (Not samplePrincipal.Identity.IsAuthenticated) Then
    ' The user is still not validated.
    MsgBox("The username and password pair is incorrect")
Else
    ' Update the current principal.
    My.User.CurrentPrincipal = samplePrincipal
    Me.Close()
End If
```

Test de l'application

Maintenant que l'application a un code d'authentification, vous pouvez l'exécuter et essayer d'authentifier un utilisateur.

Pour tester l'application

1. Démarrez l'application.
2. Cliquez sur **Authentifier**.

   Le formulaire de connexion s'ouvre.
3. Entrez **TestUser** dans la zone **Nom d'utilisateur** et **BadPassword** dans la zone **Mot de passe**, puis cliquez sur **OK**.

   Un message s'affiche et indique que la paire nom d'utilisateur/mot de passe est incorrecte.
4. Cliquez sur **OK** pour fermer le message.

5. Cliquer sur **Annuler** pour fermer le formulaire de connexion.

   Les étiquettes du formulaire principal indiquent désormais **User not authenticated** et **User is not an Administrator**.

6. Cliquez sur **Authentifier**.

   Le formulaire de connexion s'ouvre.

7. Entrez **TestUser** dans la zone de texte **Nom d'utilisateur** et **Password** dans la zone de texte **Mot de passe**, puis cliquez sur **OK**. Vérifiez que le mot de passe est entré avec le respect des majuscules.

   Les étiquettes du formulaire principal affichent désormais **Authenticated TestUser** et **User is an Administrator**.

Voir aussi

Tâches

Comment : établir une connexion à des données d'une base de données

Référence

My.User, objet

IIdentity

IPrincipal

Concepts

Accès aux données de l'utilisateur

Autres ressources

Authentification et autorisation dans le .NET Framework avec Visual Basic

Authentification et autorisation dans le .NET Framework avec Visual Basic

Classes utilisées dans les opérations d'utilisateur du .NET Framework

Les interfaces IIdentity et IPrincipal forment la base du système d'authentification utilisateur du .NET Framework. Vous pouvez utiliser les implémentations de ces interfaces fournies par le .NET Framework ou implémenter ces classes vous-même.

Classes de base utilisées pour les opérations utilisateur

La plupart des classes qui prennent en charge les opérations utilisateur figurent dans l'espace de noms System.Security.Principal. Pour une liste plus complète, consultez la documentation du .NET Framework.

Les classes suivantes implémentent l'interface **IIdentity** qui définit les fonctionnalités de base d'un objet Identity.

Classe	Description
GenericIdentity	Représente un utilisateur générique.
WindowsIdentity	Représente un utilisateur Windows.
IdentityReference	Représente une identité et constitue la classe de base des classes NTAccount et SecurityIdentifier.
NTAccount	Représente un utilisateur ou un compte de groupe.
SecurityIdentifier	Représente un identificateur de sécurité (SID) et fournit des opérations de marshaling et de comparaison des SID.

Les classes suivantes implémentent l'interface **IPrincipal** qui définit les fonctionnalités de base d'un objet Principal.

Classe	Description
GenericPrincipal	Représente une entité de sécurité générique.
WindowsPrincipal	Permet au code de vérifier l'appartenance à un groupe Windows d'un utilisateur Windows.

Ce tableau répertorie les énumérations les plus importantes utilisées pour les opérations utilisateur du .NET Framework.

Enumération	Description
PrincipalPolicy	Spécifie comment les objets Identity et Principal doivent être créés pour un domaine d'application.
WellKnownSidType	Définit un ensemble d'identificateurs de sécurité (SID) fréquemment utilisés.
WindowsAccountType	Spécifie le type de compte Windows utilisé.
WindowsBuiltInRole	Spécifie les rôles communs à utiliser avec IsInRole.

Voir aussi

Tâches

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet

Autres ressources

Authentification et autorisation dans le .NET Framework avec Visual Basic

Autorisations et sécurité d'accès au code dans le .NET Framework avec Visual Basic

Dans le .NET Framework, la sécurité d'accès de code limite l'accès du code aux ressources et aux opérations protégées. Chaque application qui cible le Common Language Runtime doit interagir avec le système de sécurité du runtime. Lorsqu'une application est exécutée, elle est automatiquement évaluée et reçoit un jeu d'autorisations de la part du runtime. En fonction des autorisations reçues, l'application peut s'exécuter correctement ou générer une exception de sécurité.

Les paramètres de sécurité locaux d'un ordinateur particulier contrôlent en fin de compte les autorisations que le code reçoit. Puisque ces paramètres peuvent varier d'un ordinateur à l'autre, vous ne pouvez jamais être sûr que votre code recevra les autorisations suffisantes pour s'exécuter. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Autorisations d'accès du code

Les objets d'autorisation d'accès du code servent à protéger les ressources et les opérations d'une utilisation par des utilisateurs non autorisés. Elles constituent un élément fondamental du mécanisme utilisé par le Common Language Runtime pour imposer des restrictions de sécurité sur le code managé.

Les autorisations d'accès du code permettent à un utilisateur d'accéder à une ressource protégée, telle qu'un fichier, ou d'exécuter une action protégée, telle que l'accès au code managé. Toutes les autorisations d'accès du code peuvent être demandées par code ; l'octroi ou non d'autorisations est déterminé par le runtime. Chaque autorisation d'accès du code provient de la classe CodeAccessPermission, et par conséquent, les autorisations ont des méthodes communes : **Assert**, **Demand**, **Deny**, **PermitOnly**, **IsSubSetOf**, **Intersec**t et **Union**.

Autorisations fournies par .NET Framework

Ce tableau affiche les autorisations d'accès du code fournies par le .NET Framework.

Classe d'autorisation	Permet
AspNetHostingPermission	Accès aux ressources dans des environnements hébergés par ASP.NET.
DirectoryServicesPermission	Accès aux classes **System.DirectoryServices**.
DnsPermission	Accès aux serveurs de système de noms de domaines (DNS) sur un réseau.
EnvironmentPermission	Lecture ou écriture dans des variables d'environnement.
EventLogPermission	Lecture ou écriture dans des services de journaux d'événements.
FileDialogPermission	Lecture ou écriture dans des fichiers au moyen d'une boîte de dialogue **Ouvrir**.
FileIOPermission	Lecture ou écriture dans des fichiers ou répertoires.
IsolatedStorageFilePermission	Lecture ou écriture dans des fichiers ou répertoires dans un stockage isolé.
MessageQueuePermission	Accès aux files d'attente de messages par le biais des interfaces MSMQ (Microsoft Message Queuing).
OdbcPermission	Accès aux sources de données ODBC (Open Database Connectivity).
OleDbPermission	Accès à des bases de données utilisant OLE DB.
OraclePermission	Accès aux bases de données Oracle.
PerformanceCounterPermission	Accès aux compteurs de performance.
PrintingPermission	Accès aux imprimantes.
ReflectionPermission	Définition d'informations se rapportant à un type au moment de l'exécution.
RegistryPermission	Lecture, écriture, création ou suppression de clés de Registre et de valeurs.
SecurityPermission	Exécution, assertion d'autorisations, appel dans du code non managé, omission de vérification et autres dr oits liés à la sécurité.

ServiceControllerPermission	Accès à des services en cours d'exécution ou arrêtés.
SocketPermission	Établissement ou acceptation de connexions sur une adresse de transport.
SqlClientPermission	Accès à des bases de données SQL.
UIPermission	Accès aux fonctionnalités de l'interface utilisateur.
WebPermission	Établissement ou acceptation de connexions sur une adresse Web.

Création de vos propres autorisations

Le .NET Framework fournit un jeu de classes d'autorisations d'accès du code conçu pour protéger un jeu spécifique de ressources et d'opérations, en s'intéressant particulièrement aux ressources exposées par le .NET Framework. Pour la plupart des environnements, les autorisations d'accès du code intégrées sont adéquates. Cependant, dans certains cas, il peut s'avérer avisé de définir votre propre classe d'autorisations d'accès du code. Pour plus d'informations, consultez Création de vos propres autorisations d'accès du code.

Autorisations d'identité

Les autorisations d'identité représentent des caractéristiques qui identifient un assembly. Le Common Language Runtime octroie des autorisations d'identité à un assembly lors de son chargement, en se basant sur les informations qu'il obtient sur celui-ci. Pour plus d'informations, consultez Autorisations d'identité.

Autorisations basées sur les rôles

Les applications destinées aux entreprises accordent souvent l'accès aux données et aux ressources en fonction des informations d'identification fournies par l'utilisateur. Pour cela, elles vérifient en général le rôle de l'utilisateur afin de lui donner accès aux ressources en rapport avec son rôle. Le Common Language Runtime fournit une prise en charge des autorisations basées sur les rôles pour des comptes Windows ou des identités personnalisées. Pour plus d'informations, consultez Sécurité basée sur les rôles.

Tâches liées à la sécurité

Le tableau suivant répertorie les tâches associées aux autorisations et à la sécurité.

Pour	Reportez-vous à
Demander l'autorisation pour accéder à du code non managé	Comment : demander l'autorisation d'accéder à du code non managé
Demander des autorisations facultatives	Comment : demander des autorisations facultatives à l'aide de l'indicateur RequestOptional
Demander l'autorisation pour un jeu d'autorisations nommé	Comment : demander l'autorisation pour un jeu d'autorisations nommé
Demande d'autorisations codées en XML	Demande d'autorisations codées en XML
Refuser des autorisations	Comment : refuser des autorisations à l'aide de l'indicateur RequestRefuse
Exécuter une vérification de sécurité impérative	Comment : exécuter les vérifications de sécurité impératives
Exécuter une vérification de sécurité déclarative	Exécution de contrôles de sécurité déclaratifs
Substituer une vérification de sécurité	Substitution des vérifications de sécurité
Partager une bibliothèque avec du code d'un niveau de confiance partiel	Partage d'une bibliothèque avec du code d'un niveau de confiance partiel
Requérir la confiance totale pour les types dans un assembly AllowPartiallyTrustedCallersAttribute	Nécessité d'une confiance totale pour des types dans un assembly AllowPartiallyTrustedCallersAttribute

Créer des objets WindowsIdentity et WindowsPrincipal.	Comment : créer un objet WindowsPrincipal
Créer des objets GenericPrincipal et GenericIdentity.	Comment : créer des objets GenericPrincipal et GenericIdentity

Modèle de stratégie de sécurité du .NET Framework

Cinq éléments composent le modèle de stratégie de sécurité du .NET Framework. Il s'agit des éléments suivants :

- Niveaux de stratégie de sécurité : entreprise, ordinateur, utilisateur et domaine d'application.
- Groupes de codes qui existent en tant que hiérarchie aux niveaux de stratégie de l'entreprise, de l'ordinateur et de l'utilisateur.
- Jeux d'autorisations nommés associés à chaque groupe de codes.
- Preuve qui fournit des informations sur l'identité du code.
- Hôtes de domaine d'application qui fournissent une preuve à propos du code au Common Language Runtime.

Niveaux de stratégie de sécurité

Quatre niveaux de sécurité sont fournis par le .NET Framework pour calculer l'octroi d'autorisations à un assembly ou à un domaine d'application. Chaque niveau contient sa propre hiérarchie de groupes de codes et ses jeux d'autorisations. Le runtime calcule le jeu d'autorisations octroyées comme la somme d'autorisations autorisée par tous les niveaux participants d'une stratégie

Ces niveaux sont les suivants :

- Stratégie d'entreprise. S'applique à tout le code managé dans un environnement d'entreprise, où un fichier de configuration d'entreprise est distribué.
- Stratégie de l'ordinateur. S'applique à tout le code managé sur l'ordinateur.
- Stratégie de l'utilisateur. S'applique au code de tous les processus associés à l'utilisateur du système d'exploitation en cours lorsque le Common Language Runtime démarre.
- Stratégie de domaine d'application. S'applique au code managé dans le domaine d'application de l'hôte.

Pour plus d'informations, consultez Niveaux de stratégie de sécurité.

Groupes de codes

Un groupe de codes est un regroupement logique de codes selon une condition particulière d'appartenance. Tout code remplissant la condition d'appartenance est inclus dans le groupe. Les groupes de codes ont associé des jeux d'autorisations qui sont évalués lors de l'établissement d'une stratégie. Pour plus d'informations, consultez Groupes de codes.

Jeux d'autorisations nommés

Un jeu d'autorisations nommé est un jeu d'autorisations que les administrateurs peuvent associer à un groupe de codes. Un jeu d'autorisations nommé est constitué d'au moins une autorisation, ainsi que d'un nom et d'une description pour le jeu. Plusieurs groupes de codes peuvent être associés ç un jeu d'autorisations spécifique.

Ce tableau présente les jeux d'autorisations nommés fournis par le common language runtime.

Nom	Description
Nothing	Pas d'autorisations (le code ne s'exécute pas).
Execution	Autorisations d'exécuter, mais pas d'autorisations d'utiliser des ressources protégées.
Internet	Jeu d'autorisations de la stratégie par défaut adaptée au contenu d'origine inconnue.
Local Intranet	Jeu d'autorisations de la stratégie par défaut au sein d'une entreprise.
Everything	Toutes autorisations standard (intégrées), à l'exception de l'autorisation d'ignorer la vérification.
FullTrust	Accès intégral à toutes les ressources.

Pour plus d'informations, consultez Jeux d'autorisations nommés.

Preuve

La preuve est l'information que le Common Language Runtime utilise pour prendre des décisions fondées sur la stratégie de sécurité. La preuve indique au runtime que le code a une caractéristique particulière et peut inclure le répertoire de l'application, l'éditeur, le site et

l'URL. Pour plus d'informations, consultez Preuve.

Hôtes de domaine d'application

Chaque application .NET Framework s'exécute dans un domaine d'application sous le contrôle d'un hôte qui crée le domaine d'application et y charge les assemblys. Les domaines d'application peuvent inclure :

- Les hôtes de navigateur. Exécutent le code dans le contexte d'un site Web.
- Les hôtes conçus de manière personnalisée. Créent des domaines et chargent des assemblys dans les domaines, y compris des assemblys dynamiques.
- Les hôtes de serveur, exécutent le code qui gère les demandes envoyées à un serveur.
- Les hôtes de shell. Lancent des applications (fichiers .exe) à partir du shell.

Ce tableau répertorie les tâches associées aux domaines d'application :

Pour	Reportez-vous à
Créer un domaine d'application.	Comment : créer un domaine d'application
Définir la stratégie de sécurité de niveau de domaine d'application	Définition d'une stratégie de sécurité au niveau du domaine d'application
Configurer un domaine d'application.	Comment : configurer un domaine d'application
Appeler des fonctions dans un domaine d'application spécifique	Calling Functions in a Specific Application Domain
Récupérer des informations d'installation à partir d'un domaine d'application	Récupération d'informations d'installation à partir d'un domaine d'application
Décharger un domaine d'application	Comment : décharger un domaine d'application

Pour plus d'informations, consultez AppDomain, Hôtes de domaine d'application et Programmation avec des domaines d'application.

Voir aussi

Tâches

Dépannage des exceptions de sécurité d'accès du code

Concepts

Isolation par utilisateur, par domaine et par assembly

Autres ressources

Configuration de la stratégie de sécurité à l'aide de l'outil .NET Framework Configuration Tool (Mscorcfg.msc)

Configuration de la stratégie de sécurité à l'aide de l'outil Code Access Security Policy Tool (Caspol.exe)

Collections dans le .NET Framework avec Visual Basic

Le .NET Framework fournit des classes spécialisées pour le stockage et la récupération des données. Ces classes fournissent la prise en charge des piles, files d'attente, listes et tables de hachage. La plupart des classes de collection implémentent les mêmes interfaces. Vous pouvez créer vos propres classes de collection, si nécessaire, en implémentant ces interfaces.

Vous devez déterminer quel type de collection est le plus adapté à vos besoins.

Différences entre les classes de collection Visual Basic et .NET Framework

Les classes de collection du .NET Framework sont définies dans les espaces de noms System.Collections, System.Collections.Generic, System.Collections.Specialized et System.Collections.ObjectModel. Les principales différences entre les classes de collection Visual Basic et du .NET Framework sont les suivantes :

- **Base d'index**. Les collections du .NET Framework sont de base zéro, tandis que la collection Visual Basic est de base 1. Cela signifie que les éléments d'une collection Visual Basic ont des valeurs d'index allant de 1 à la valeur du Count, propriété (objet Collection), alors que les éléments d'une collection du .NET Framework ont des valeurs d'index allant de 0 à une valeur inférieure à la propriété Count de la collection.

- **Type d'élément**. La collection Visual Basic prend en charge les éléments de type Object. Cette collection n'est pas de type sécurisé : vous pouvez ajouter un élément à n'importe quel type de données. Cela se traduit généralement par des performances diminuées parce que vous devez convertir les éléments du type de données **Object** vers leur type réel.

  Certaines des collections du .NET Framework ont également des éléments de type **Object**, mais beaucoup d'autres sont fortement typées, ce qui signifie qu'elles prennent en charge des éléments d'un type spécifique qui les sécurise et qui optimise les performances.

- **Éléments à clé**. La collection Visual Basic vous permet de spécifier une clé lorsque vous y ajoutez un élément. La clé est une valeur de chaîne unique que vous pouvez utiliser ultérieurement pour accéder à cet élément particulier. Les collections du .NET Framework varient par rapport aux clés. Certaines prennent en charge les clés, d'autres non.

Classes System.Collections

Les classes de l'espace de noms **System.Collections** ne stockent pas les éléments comme des objets spécifiquement typés, mais comme des objets de type **Object**.

Le tableau suivant décrit certaines des classes fréquemment utilisées :

Classe	Description
ArrayList	Implémente l'interface IList à l'aide d'un tableau dont la taille est augmentée de manière dynamique.
BitArray	Gère un tableau compact de valeurs de bit qui sont représentées par des valeurs **Boolean**, ou **True** indique que le b it est activé (1) et **False** que le bit est désactivé (0).
Hashtable	Représente une collection de paires clé/valeur qui sont organisées en fonction du code de hachage de la clé.
Queue	Représente une collection d'objets de type premier entré, premier sorti (FIFO, first-in-first-out).
Stack	Représente une simple collection d'objets non générique de type dernier entré, premier sorti (LIFO, last-in-first-out).

Classes System.Collections.Generic et System.Collections.ObjectModel

Les espaces de noms **System.Collections.Generic** et **System.Collections.ObjectModel** fournissent des types génériques qui vous permettent de créer des collections fortement typées et de spécifier le type de données de l'élément lorsque vous les créez.

Le tableau suivant décrit certaines des classes fréquemment utilisées :

Classe	Description
Collection	Fournit la classe de base pour une collection générique.

Dictionary	Représente une collection de paires clé/valeur organisées en fonction de la clé.
KeyedCollection	Fournit la classe de base abstraite pour une collection dont les clés sont incorporées dans les valeurs.
LinkedList	Représente une liste à lien double.
LinkedListNode	Représente un nœud dans un **LinkedList**. Cette classe ne peut pas être héritée.
List	Implémente l'interface IList à l'aide d'un tableau dont la taille est augmentée de manière dynamique.
Queue	Représente une collection d'objets de type premier entré, premier sorti.
SortedDictionary	Représente une collection de paires clé/valeur triées sur la clé.
SortedList	Représente une collection de paires clé/valeur triées par clé en fonction de l'implémentation IComparer as sociée.
Stack	Représente une collection d'instances à taille variable de type dernier entré, premier sorti (LIFO) du même type arbitraire.
ReadOnlyCollection	Fournit la classe de base pour une collection en lecture seule générique.

Classes System.Collections.Specialized

L'espace de noms **System.Collections.Specialized** fournit des classes de collection spécialisées et fortement typées, telles que les collections à chaîne unique et les dictionnaires de listes liées et hybrides.

Le tableau suivant décrit certaines des classes fréquemment utilisées :

Classe	**Description**
CollectionsUtil	Crée des collections qui ne tiennent pas compte des majuscules et des minuscules dans les chaînes .
HybridDictionary	Implémente IDictionary à l'aide d'un ListDictionary lorsque la collection est petite, puis passe à un **Hashtable** lorsque la collection s'agrandit.
ListDictionary	Implémente **IDictionary** à l'aide d'une liste liée unique. Recommandée pour les collections qui co ntiennent généralement 10 éléments au plus.
NameObjectCollectionBase	Fournit la classe de base abstraite pour une collection de clés de chaîne et de valeurs d'objet associ ées accessibles à l'aide de la clé ou de l'index.
NameValueCollection	Représente une collection de clés de chaîne et de valeurs de chaîne associées accessibles à l'aide d e la clé ou de l'index.
OrderedDictionary	Représente une collection de paires clé/valeur classées en fonction de la clé ou de l'index.
StringCollection	Représente une collection de chaînes.
StringDictionary	Implémente un tableau de hachage à l'aide de la clé et de la valeur fortement typées pour être des chaînes plutôt que des objets.

Voir aussi

Référence

Collection, objet (Visual Basic)

Concepts

Collections en Visual Basic
Autres ressources
Fonctionnalités plus poussées de Visual Basic

E/S de fichier dans le .NET Framework avec Visual Basic

Lors de l'exécution d'opérations d'E/S de fichier complexes, telles que le stockage isolé, les utilisateurs de Visual Basic peuvent être obligés d'étudier le .NET Framework et les classes qu'il offre. Cette section fournit une vue d'ensemble des concepts **FileSystem** et de l'E/S de fichier dans le .NET Framework et une liste de classes couramment utilisées.

Dans cette section

Concepts de base du système de fichiers et des E/S de fichier du .NET Framework

Fournit une vue d'ensemble des concepts E/S de fichier dans le .NET Framework, à savoir les flux, le stockage isolé, les événements de fichier, les attributs de fichier et l'accès aux fichiers.

Classes utilisées dans les E/S de fichier du .NET Framework et le système de fichiers

Répertorie les classes utilisées dans l'E/S de fichier du .NET Framework et du **FileSystem**, y compris les classes E/S de base, les classes qui créent des flux et les classes qui créent des writers et des lecteurs de flux.

Voir aussi

Tâches

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)
Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic
Comment : lire des fichiers texte dans Visual Basic

Référence

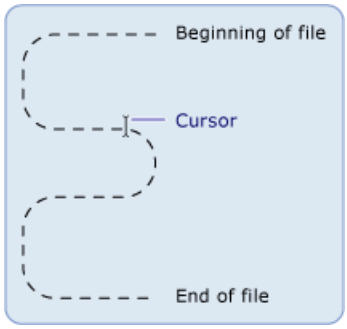
System.IO Namespace

Concepts de base du système de fichiers et des E/S de fichier du .NET Framework

L'espace de noms System.IO contient les classes File et Directory, qui offrent la fonctionnalité du .NET Framework permettant de manipuler des fichiers et des répertoires. Étant donné que toutes les méthodes de ces objets sont des membres statiques ou partagés, vous pouvez les utiliser directement sans créer d'instance de classe au préalable. Les classes FileInfo et DirectoryInfo sont associées à ces classes qui seront familières aux utilisateurs de la fonctionnalité **My**. Pour utiliser ces interfaces, vous devez qualifier pleinement les noms ou importer les espaces de noms appropriés en incluant les instructions **Imports** au début du code affecté. Pour plus d'informations, consultez Imports, instruction.

Définition d'un flux

Le .NET Framework utilise des flux pour prendre en charge la lecture et l'écriture dans les fichiers. Vous pouvez considérer un flux comme un ensemble de données contiguës unidimensionnel qui a un début et une fin et dans lequel le curseur indique la position actuelle dans le flux.



Opérations de flux

Les données contenues dans le flux peuvent provenir de la mémoire, d'un fichier ou d'un socket TCP/IP. Il existe des opérations fondamentales qui peuvent s'appliquer aux flux :

- Lecture. Vous pouvez lire un flux, en transférant les données du flux vers une structure de données, par exemple une chaîne ou un tableau d'octets.
- **Écriture**. Vous pouvez écrire dans un flux, en transférant des données d'une source de données aux flux.
- **Recherche**. Vous pouvez interroger et modifier votre position dans le flux.

Pour plus d'informations, consultez Composition de flux.

Types de flux

Dans le .NET Framework, un flux est représenté par la classe Stream qui forme la classe abstraite de tous les autres flux. Vous ne pouvez pas créer directement une instance de la classe **Stream**, mais vous devez utiliser l'une des classes qu'elle implémente.

Il existe de nombreux types de flux, mais pour les besoins de l'utilisation de l'entrée/sortie (E/S) de fichier, les types les plus importants sont la classe FileStream, qui permet de lire et d'écrire dans les fichiers, et la classe IsolatedStorageFileStream qui permet de créer des fichiers et des répertoires dans un stockage isolé. Les autres flux qui peuvent être utilisés lors de l'utilisation de l'E/S de fichier sont les suivants :

- BufferedStream
- CryptoStream
- MemoryStream
- NetworkStream.

Le tableau suivant répertorie les tâches couramment accomplies avec un flux :

Pour	Reportez-vous à

Lire et écrire dans un fichier de données	Comment : lire et écrire dans un fichier de données créé récemment
Lire dans un fichier XML	Tâches courantes de XmlReader
Créer un writer de flux afin d'écrire du texte dans un fichier	Création d'un writer
Lire du texte à partir d'un fichier	Comment : lire du texte dans un fichier
Écrire du texte dans un fichier	Comment : écrire du texte dans un fichier
Lire des caractères à partir d'une chaîne	Comment : lire les caractères d'une chaîne
Écrire des caractères dans une chaîne	Comment : écrire des caractères dans une chaîne
Chiffrer des données	Chiffrement de données
Déchiffrer des données	Déchiffrement de données

Accès et attributs de fichier

Vous pouvez contrôler comment les fichiers sont créés, ouverts, et partagés à l'aide des énumérations FileAccess, FileMode et FileShare qui contiennent les indicateurs utilisés par les constructeurs de la classe **FileStream**. Par exemple, lorsque vous ouvrez ou créez un **FileStream**, l'énumération **FileMode** vous permet de spécifier si le fichier est ouvert pour permettre l'ajout, si un fichier est créé lorsque le fichier spécifié n'existe pas, si le fichier est remplacé, etc.

L'énumération FileAttributes permet la collecte d'informations spécifiques au fichier. L'énumération **FileAttributes** retourne les attributs stockés du fichier suivants : compressé, chiffré, caché, en lecture seule, archive, répertoire, fichier système ou fichier temporaire.

Le tableau suivant répertorie les tâches relatives à l'accès et aux attributs de fichier :

Pour	**Reportez-vous à**
Ouvrir et ajouter du texte à un fichier journal	Comment : ouvrir un fichier journal et y ajouter des éléments
Déterminer les attributs d'un fichier	**FileAttributes**

Autorisations de fichier

À l'aide de la classe FileIOPermission, vous pouvez contrôler l'accès aux fichiers et aux répertoires. Cette fonctionnalité peut s'avérer particulièrement importante pour les développeurs qui emploient des formulaires Web (Web Forms). Par défaut, ceux-ci s'exécutent dans le contexte du compte d'utilisateur local ASPNET, créé dans le cadre de l'installation de ASP.NET et du .NET Framework. Lorsqu'une telle application demande l'accès à une ressource, le compte d'utilisateur ASPNET a des autorisations limitées qui peuvent empêcher l'utilisateur d'exécuter des actions telles que l'écriture dans un fichier à partir d'une application Web. Pour plus d'informations, consultez Autorisations de sécurité, Demande d'autorisations et FileIOPermission, classe.

Le tableau suivant répertorie les tâches relatives aux autorisations de fichier.

Pour	**Reportez-vous à**
Demander des autorisations minimales	Comment : demander l'autorisation d'accéder à du code non managé
Demander des autorisations facultatives	Comment : demander des autorisations facultatives à l'aide de l'indicateur RequestOptional
Refuser des autorisations	Comment : refuser des autorisations à l'aide de l'indicateur RequestRefuse

Demander des autorisations intégrées	Comment : demander l'autorisation pour un jeu d'autorisations nommé
Demander des autorisations codées en XML	Demande d'autorisations codées en XML

Stockage de fichiers isolé

Le stockage isolé est une tentative de résoudre les problèmes survenus lors de l'utilisation des fichiers alors que l'utilisateur ou le code n'a peut-être pas les autorisations nécessaires. Le stockage isolé assigne à chaque utilisateur un compartiment de données qui peut contenir plusieurs magasins. Les magasins peuvent être isolés les uns des autres par utilisateur et par assembly. Seul l'utilisateur ou l'assembly qui a créé un magasin peut y accéder. Un magasin se comporte comme un système de fichiers virtuel : dans un magasin, vous pouvez créer et manipuler des répertoires et des fichiers.

Le tableau suivant répertorie les tâches couramment associées au stockage de fichiers isolé.

Pour	**Reportez-vous à**
Créer un magasin isolé	Obtention de magasins
Énumérer des magasins isolés	Énumération de magasins
Supprimer un magasin isolé	Suppression de magasins
Créer un fichier ou un répertoire dans un stockage isolé	Création de fichiers et de répertoires
Rechercher un fichier dans un stockage isolé	Recherche de fichiers et de répertoires existants
Lire ou écrire dans un fichier dans un stockage isolé	Lecture et écriture dans des fichiers
Supprimer un fichier ou un répertoire dans un stockage isolé	Suppression de fichiers et de répertoires

Événements de fichier

Le composant FileSystemWatcher vous permet de surveiller les modifications dans les fichiers et répertoires de votre système ou d'un ordinateur auquel vous avez un accès réseau. Par exemple, si un fichier est modifié, vous pouvez envoyer une alerte à un utilisateur pour lui signaler que la modification a eu lieu. Lorsque des modifications se produisent, un ou plusieurs événements sont déclenchés, stockés dans une mémoire tampon et transmis au composant **FileSystemWatcher** pour traitement. Pour plus d'informations, consultez Réponse à des événements de système de fichiers.

Pour	**Reportez-vous à**
Créer un gestionnaire pour les événements du système de fichiers	Comment : créer des gestionnaires pour les événements de système de fichiers
Configurer une instance du composant **FileSystemWatcher**	Comment : configurer des instances du composant FileSystemWatcher

Voir aussi

Concepts

Composition de flux
E/S de fichier de base
E/S sur fichier asynchrones
Classes utilisées dans les E/S de fichier du .NET Framework et le système de fichiers

Autres ressources

Stockage isolé

Classes utilisées dans les E/S de fichier du .NET Framework et le système de fichiers

Les tableaux suivants répertorient les classes couramment utilisées pour l'E/S de fichier du .NET Framework, les classes par catégorie dans l'E/S de fichier, les classes utilisées pour créer des flux et les classes utilisées pour lire et écrire dans les flux.

Pour consulter la documentation du .NET Framework 2.0 et rechercher une liste plus complète, consultez Vue d'ensemble de la bibliothèque de classes du .NET Framework.

Classes E/S de base pour les fichiers, les lecteurs et les répertoires

Le tableau suivant répertorie et décrit les classes principales utilisées pour l'E/S de fichier.

Classe	Description
System.IO.Directory	Fournit des méthodes statiques pour la création, le déplacement et l'énumération dans les répertoires et les sous-répertoires.
System.IO.DirectoryInfo	Fournit des méthodes d'instance pour la création, le déplacement et l'énumération dans les répertoires et les sous-répertoires.
System.IO.DriveInfo	Fournit des méthodes d'instance pour la création, le déplacement et l'énumération dans les lecteurs.
System.IO.File	Fournit des méthodes statiques pour la création, la copie, la suppression, le déplacement et l'ouverture de fichiers et permet de créer un **FileStream**.
System.IO.FileAccess	Définit des constantes pour un accès en lecture, en écriture ou en lecture/écriture à un fichier.
System.IO.FileAttributes	Fournit des attributs pour les fichiers et les répertoires tels que Archive, Hidden et ReadOnly.
System.IO.FileInfo	Fournit des méthodes statiques pour la création, la copie, la suppression, le déplacement et l'ouverture de fichiers et permet de créer un **FileStream**.
System.IO.FileMode	Contrôle comment un fichier est ouvert. Ce paramètre est spécifié dans de nombreux constructeurs pour **FileStream** et **IsolatedStorageFileStream** ainsi que pour les méthodes **Open** de **File** et de **FileInfo**.
System.IO.FileShare	Définit des constantes pour le contrôle du type d'accès que d'autres flux de fichiers peuvent avoir sur le même fichier.
System.IO.Path	Fournit des méthodes et des propriétés pour le traitement des chaînes de répertoire.
System.Security.Permissions.FileIOPermission	Contrôle l'accès des fichiers et des dossiers en définissant les autorisations Read, Write, Append et PathDiscovery.

Classes utilisées pour créer des flux

Le tableau suivant répertorie et décrit les classes principales utilisées pour créer des flux.

Classe	Description
System.IO.BufferedStream	Ajoute une couche de mise en mémoire tampon aux opérations de lecture et d'écriture sur un autre flux.

System.IO.FileStream	Prend en charge l'accès aléatoire aux fichiers via sa méthode Seek. **FileStream** ouvre les fichiers de manière synchrone par défaut mais il prend également en charge des opérations asynchrones.
System.IO.MemoryStream	Crée un flux dont le magasin de stockage est une mémoire et non un fichier.
System.Net.Sockets.NetworkStream	Fournit le flux de données sous-jacent pour l'accès réseau.
System.Security.Cryptography.CryptoStream	Définit un flux qui lie les flux de données aux transformations de chiffrement.

Classes utilisées pour lire et écrire dans des flux

Le tableau suivant affiche les classes spécifiques utilisées pour lire et écrire dans des fichiers avec des flux.

Classe	Description
System.IO.BinaryReader	Lit des chaînes codées et des types de données primitifs à partir d'un **FileStream**.
System.IO.BinaryWriter	Écrit des chaînes codées et des types de données primitifs dans un **FileStream**.
System.IO.StreamReader	Lit des caractères dans un **FileStream**, à l'aide de CurrentEncoding pour convertir des caractères en octets et vice versa. **StreamReader** a un constructeur qui essaie de vérifier le **CurrentEncoding** correct d'un flux donné, en fonction de la présence d'un préambule spécifique à **CurrentEncoding**, tel qu'une marque d'ordre d'octet.
System.IO.StreamWriter	Écrit des caractères dans un **FileStream**, à l'aide de Encoding pour convertir des caractères en octets.
System.IO.StringReader	Lit des caractères dans **String**. La sortie peut être un flux de n'importe quel codage ou un **String**.
System.IO.StringWriter	Écrit des caractères dans un **String**. La sortie peut être un flux de n'importe quel codage ou un **String**.

Voir aussi

Concepts

Composition de flux
E/S de fichier de base
E/S sur fichier asynchrones
Concepts de base du système de fichiers et des E/S de fichier du .NET Framework

Autres ressources

Stockage isolé

Graphiques dans le .NET Framework avec Visual Basic

Le .NET Framework fournit l'interface de programmation d'applications (API, Application Programming Interface) GDI+ pour la manipulation de graphiques. GDI+ est une implémentation avancée de Graphics Device Interface (Interface de périphériques graphiques) de Windows (GDI). Grâce à GDI+, vous pouvez créer des graphiques, dessiner du texte et manipuler des images graphiques en tant qu'objets.

GDI+ est une interface conçue pour allier performances et simplicité d'utilisation. Vous pouvez utiliser GDI+ en vue du rendu des images graphiques sur des Windows Forms et des contrôles. Bien qu'il soit impossible d'utiliser GDI+ directement dans les Web Forms, vous pouvez afficher des images graphiques par l'intermédiaire du contrôle serveur Web Image.

Présentation de GDI+

Lorsque vous créez un contrôle Windows Forms, vous pouvez utiliser GDI+ pour accéder à son image et le mettre à jour. Vous pouvez également utiliser GDI+ pour créer vos propres images, indépendantes de l'interface utilisateur de votre application.

Pour dessiner une image dans le .NET Framework, vous devez utiliser l'objet Graphics associé à l'image.

Dans certains cas, vous pouvez obtenir directement l'objet **Graphics** de l'image. Par exemple, lorsque vous créez un contrôle Windows Forms, vous pouvez substituer la méthode OnPaint pour accéder à l'objet **Graphics** de l'image du contrôle.

Dans d'autres cas, par exemple lorsque vous créez votre propre image, vous devez également créer un objet graphique. La méthode FromImage partagée prend une image et retourne un objet **Graphics** associé à cette image.

La classe **Graphics** contient plusieurs méthodes de dessin et de manipulation d'images. Certaines de ces méthodes couramment utilisées sont répertoriées ci-dessous :

- Méthodes pour dessiner des lignes : DrawArc, DrawBezier, DrawEllipse, DrawImage, DrawLine, DrawPolygon, DrawRectangle et DrawString.
- Méthodes pour remplir des formes : FillClosedCurve, FillEllipse, FillPath, FillPolygon et FillRectangle.
- Méthode pour effacer la surface de dessin : Clear.
- Méthode pour créer un objet **Graphics** à partir d'une image : **FromImage**.

De nombreuses méthodes répertoriées ci-dessus de prennent comme arguments des structures ou des classes définies dans l'espace de noms System.Drawing. Le tableau suivant répertorie certaines des classes et structures GDI+ les plus utilisées.

Classe/structure	Description
System.Drawing.Bitmap	Encapsule une bitmap GDI+ qui est composée des données de pixels d'une image graphique et de ses attributs. Un objet **Bitmap** est l'objet utilisé pour manipuler des images définies par des données de pixels.
System.Drawing.Brushes	Définit des pinceaux pour toutes les couleurs standard.
System.Drawing.Color	Représente une couleur ARVB.
System.Drawing.Font	Définit un format spécifique pour le texte, notamment, la police, la taille et les attributs de style.
System.Drawing.Pen	Définit un objet utilisé pour dessiner des lignes et des courbes.
System.Drawing.Pens	Définit des stylets pour toutes les couleurs standard.
System.Drawing.Point	Représente une paire ordonnée de coordonnées x et y entières définissant un point dans un pla n à deux dimensions.
System.Drawing.Rectangle	Stocke un ensemble de quatre entiers qui représentent l'emplacement et la taille d'un rectangle. Pour des fonctions de région plus avancées, utilisez un objet Region.

System.Drawing.SolidBrush	Définit un pinceau d'une seule couleur. Les pinceaux sont utilisés pour remplir les formes des gr aphiques, telles que les rectangles, les ellipses, les secteurs, les polygones et les tracés.
System.Drawing.TextureBrush	Chaque propriété de la classe **TextureBrush** est un objet Brush qui utilise une image pour rem plir l'intérieur d'une forme.

Gestion des ressources

De nombreuses classes de dessin implémentent IDisposable parce qu'elles encapsulent des ressources système non managées. Si vous créez une instance de l'une de ces classes, vous devez appeler la méthode Dispose de la classe lorsque vous avez fini avec l'objet.

Vous pouvez également créer l'objet avec l'instruction **Using** qui appelle implicitement la méthode **Dispose** de l'objet. Pour plus d'informations, consultez Durée de vie d'un objet : création et destruction des objets et Using, instruction (Visual Basic).

Rubriques connexes

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Décrit les modifications apportées au modèle de rendu de graphiques dans Visual Basic 2005.

Graphiques et dessins dans les Windows Forms

Documentation pour l'utilisation des graphiques dans les applications Windows Forms.

Vue d'ensemble des graphismes (Windows Forms)

Fournit une présentation des classes managées relatives aux graphiques.

À propos du code managé GDI+

Fournit des informations concernant les classes GDI+ managées.

Utilisation de classes graphiques managées

Montre comment exécuter diverses tâches à l'aide des classes managées GDI+.

Peinture et rendu personnalisés des contrôles

Explique en détail comment fournir le code destiné à peindre des contrôles.

Vue d'ensemble du contrôle serveur Web Image

Décrit le contrôle vous permettant d'afficher des images sur une page Web Forms et de les gérer à l'aide de code.

Éditeur d'images

Fournit des liens vers des rubriques traitant de l'utilisation de l'éditeur d'images pour créer des fichiers image qui seront utilisés dans une application.

Voir aussi

Référence

Using, instruction (Visual Basic)

System.Drawing

Concepts

Durée de vie d'un objet : création et destruction des objets

Enregistrement et traçage dans le .NET Framework avec Visual Basic

Les objets **My.Application.Log** et **My.Log** reposent sur les classes d'enregistrement du .NET Framework. Lorsque vous devez personnaliser les objets **Log**, en ajoutant de nouveaux écouteurs ou en filtrant la sortie de journal, vous utilisez le .NET Framework. Cette section propose une vue d'ensemble des concepts de traçage et de débogage dans le .NET Framework et une liste de classes couramment utilisées.

Dans cette section

Concepts de base des journaux du .NET Framework

Fournit une vue d'ensemble des concepts d'enregistrement, de traçage et d'instrumentation dans le .NET Framework.

Classes utilisées dans le .NET Framework pour les enregistrements dans le fichier journal

Répertorie les classes utilisées dans l'enregistrement, le traçage et l'instrumentation du .NET Framework.

Voir aussi

Référence

My.Application.Log, objet

My.Log, objet

System.Diagnostics

Concepts

Enregistrement d'informations provenant de l'application

Autres ressources

Traçage et instrumentation d'applications

Concepts de base des journaux du .NET Framework

Outre l'utilisation des objets **My.Application.Log** et **My.Log**, il existe de nombreuses méthodes pour enregistrer les informations de vos applications Visual Basic. De nombreuses classes .NET Framework sont disponibles. En enregistrant les informations, vous pouvez contrôler et mesurer les performances de votre application et diagnostiquer les erreurs.

Traçage, débogage et instrumentation

La documentation .NET Framework utilise plusieurs termes différents relatifs à l'enregistrement : traçage, débogage et instrumentation.

- Le *traçage* vous permet de contrôler l'exécution de votre application lorsque celle-ci s'exécute. Il est généralement implémenté à l'aide de la classe Trace. Vous pouvez ajouter des instructions de traçage à votre application .NET lors du développement. Vous pouvez les utiliser lors du développement et après les avoir déployées.
- Le *débogage* est similaire au traçage, mais il est en général plus documenté et est désactivé dans les versions release. Il est généralement implémenté à l'aide de la classe **Trace**.
- L'*instrumentation* fait référence à l'ajout de code à votre application contrôlant ou mesurant le niveau de performance d'un produit. L'enregistrement, le traçage, et le débogage sont tous des formes d'instrumentation. Vous pouvez instrumenter votre application en plaçant des instructions de traçage à des emplacements stratégiques dans votre code. Cette méthode est particulièrement utile pour les applications distribuées.

  Grâce aux instructions de traçage, vous pouvez instrumenter une application pour afficher des informations en cas de problème et contrôler son niveau d'exécution.

Les classes **Trace** et Debug peuvent, à l'instar des objets **Log** Visual Basic, enregistrer les informations relatives aux erreurs et à l'exécution de l'application dans des journaux, des fichiers texte ou d'autres périphériques en vue d'une analyse ultérieure. En outre, elles proposent des méthodes qui permettent un contrôle plus fin de la sortie. Pour obtenir plus de classes de traçage, consultez Classes utilisées dans le .NET Framework pour les enregistrements dans le fichier journal.

Notions de base du journal des événements et du traçage

Une fois que vous instrumentez votre application, vous pouvez :

- **tracer le code ;** recevoir des messages informatifs sur l'exécution d'une application au moment de l'exécution. Pour plus d'informations, consultez Introduction à l'instrumentation et au traçage ;
- **recevoir la sortie de débogage ;** détecter et résoudre les erreurs de programmation dans une application en cours de développement. Pour plus d'informations, consultez Débogage dans Visual Studio ;
- **Accès aux compteurs de performance.** suivre les performances de votre application. Pour plus d'informations, consultez Introduction à l'analyse des seuils de performance ;
- **écrire dans le journal des événements ;** suivre les événements majeurs dans l'exécution de votre application. Pour plus d'informations, consultez Journalisation des événements d'application, de serveur et de sécurité.

Les classes **Trace** et **Debug** vous permettent de contrôler et d'examiner les performances d'une application lors du développement ou après le déploiement. Par exemple, vous pouvez utiliser la classe **Trace** pour suivre des actions particulières dans une application déployée au fur et à mesure qu'elles se produisent (par exemple, la création de nouvelles connexions de base de données) et ainsi contrôler l'efficacité de l'application.

Voir aussi

Référence

System.Diagnostics

Concepts

Enregistrement et traçage dans le .NET Framework avec Visual Basic
Introduction à l'instrumentation et au traçage
Classes utilisées dans le .NET Framework pour les enregistrements dans le fichier journal

Autres ressources

Traçage et instrumentation d'applications

Classes utilisées dans le .NET Framework pour les enregistrements dans le fichier journal

Cette rubrique contient des tableaux qui répertorient les classes couramment utilisées pour l'enregistrement et le traçage du .NET Framework. Les tableaux sont divisés en classes de sortie de journal, d'écouteur de journalisation et de filtrage de journal. Pour consulter la documentation du .NET Framework 2.0 et rechercher une liste plus complète, consultez Vue d'ensemble de la bibliothèque de classes du .NET Framework.

Classes de base pour la sortie de journal

Le tableau suivant répertorie et décrit les classes principales utilisées pour la sortie de journal.

Classe	Description
Debug	Fournit un ensemble de méthodes et de propriétés qui permettent de déboguer votre code. Cette classe ne peut pas être héritée.
Trace	Fournit un ensemble de méthodes et de propriétés qui vous permettent de tracer l'exécution de votre code. Cette classe ne peut pas être héritée.
TraceSource	Fournit un ensemble de méthodes et de propriétés qui vous permettent de tracer l'exécution de votre code.

Classes d'écouteur de journalisation

Le tableau suivant répertorie et décrit les principales classes de l'écouteur de journalisation.

Classe	Description
TraceListener	Fournit la classe de base abstraite pour les écouteurs qui contrôlent la sortie de trace et de débogage.
ConsoleTraceListener	Dirige la sortie de traçage ou de débogage vers la sortie standard ou le flux d'erreurs standard.
DefaultTraceListener	Fournit les méthodes de sortie et le comportement par défaut du traçage.
DelimitedListTraceListener	Dirige la sortie de traçage ou de débogage vers un TextWriter, tel que StreamWriter, ou vers un Stream, tel que FileStream.
EventLogTraceListener	Fournit un écouteur simple qui dirige la sortie de traçage ou de débogage vers un EventLog.
FileLogTraceListener	Fournit un écouteur simple qui dirige la sortie d'enregistrement vers un fichier.
TextWriterTraceListener	Dirige la sortie de traçage ou de débogage vers un **TextWriter** ou un **Stream** tel que Out ou **FileStream**.
XmlWriterTraceListener	Dirige la sortie de traçage ou de débogage vers un **TextWriter** ou un **Stream** tel que **FileStream**.

Filtrage de journal et basculement de classes

Le tableau suivant répertorie et décrit le filtrage de journal principal et le basculement de classes.

Classe	Description
Switch	Fournit une classe de base abstraite pour créer des commutateurs de débogage et de traçage.
BooleanSwitch	Fournit un commutateur simple activé/désactivé qui contrôle la sortie de débogage et de traçage.

SourceSwitch	Fournit un commutateur à plusieurs niveaux pour contrôler la sortie de traçage et de débogage sans recompiler votre code.
TraceSwitch	Fournit un commutateur à plusieurs niveaux pour contrôler la sortie de traçage et de débogage sans recompiler votre code.
EventTypeFilter	Fournit un commutateur basé sur le niveau de gravité et qui contrôle la sortie de traçage.
SourceFilter	Spécifie si un écouteur doit tracer un message en fonction de la source d'une trace.
TraceFilter	Définit une méthode pour les implémentations du filtre de trace.

Voir aussi

Référence

System.Diagnostics

Concepts

Enregistrement et traçage dans le .NET Framework avec Visual Basic

Concepts de base des journaux du .NET Framework

Autres ressources

Traçage et instrumentation d'applications

Opérations de réseau dans le .NET Framework avec Visual Basic

Lors de l'exécution d'opérations de réseau complexes, les utilisateurs Visual Basic devront éventuellement explorer le .NET Framework et les classes qu'il propose. Cette section propose une vue d'ensemble des concepts de réseau dans le .NET Framework et une liste de classes couramment utilisées.

Dans cette section

Concepts de base des opérations de réseau du .NET Framework

Propose une vue d'ensemble des concepts de réseau dans le .NET Framework, y compris des classes WebRequest et WebResponse, de la sécurité et du traçage réseau.

Classes utilisées dans les opérations de réseau du .NET Framework

Répertorie les classes utilisées dans les opérations de réseau du .NET Framework.

Référence

WebRequest

Effectue une demande à un URI (Uniform Resource Identifier).

WebResponse

Fournit une réponse d'un URI.

WebClient

Fournit des méthodes communes pour l'envoi de données à une ressource identifiée par un URI ou pour la réception de données en provenance de cette ressource.

Concepts de base des opérations de réseau du .NET Framework

Le Microsoft .NET Framework fournit une implémentation en couche des services Internet, extensible et managée, qui peut être rapidement et facilement intégrée à vos applications. Pour utiliser ces classes, vous devez qualifier pleinement les noms ou importer les espaces de noms appropriés en incluant une ou plusieurs instructions **Imports** au début du code affecté. Pour plus d'informations, consultez Imports, instruction.

Accès aux ressources Internet

Pour accéder aux ressources Internet via un modèle demande/réponse, trois éléments d'information sont requis. Ils sont fournis par des classes du .NET Framework spécifiques :

- La classe Uri contient l'URI d'une ressource Internet qui se compose au moins de trois, voire quatre parties : l'identificateur de modèle qui identifie le protocole de communication ; l'identificateur de serveur ; l'identificateur de chemin d'accès ; et enfin, une chaîne de requête facultative.
- La classe WebRequest contient une demande pour la ressource.
- La classe WebResponse fournit un conteneur pour la réponse entrante.

Transfert et téléchargement de données

Pour les applications qui ont besoin de faire des demandes simples de ressources Internet, la classe WebClient fournit des méthodes de transfert ou de téléchargement des données. Étant donné que **WebClient** s'appuie sur la classe **WebRequest** pour fournir l'accès aux ressources Internet, il peut utiliser n'importe quel protocole enfichable inscrit.

L'espace de noms System.Net.Sockets fournit les classes TcpClient, TcpListener et UdpClient pour les applications qui n'utilisent pas le modèle demande/réponse ou pour celles qui ont besoin d'écouter sur le réseau. Ces classes gèrent les détails de l'établissement des connexions à l'aide de différents protocoles de transport et exposent la connexion réseau à l'application sous la forme d'un flux.

Pour plus d'informations, consultez Demande de données.

Le tableau suivant répertorie des tâches courantes qui impliquent la demande de données à partir d'une ressource Internet.

Pour	**Reportez-vous à**
Créer une demande Internet	Création de demandes Internet
Envoyer des données au réseau	Utilisation de flux sur le réseau
Effectuer des demandes asynchrones pour une ressource Internet	Demandes asynchrones
Accéder aux propriétés HTTP	HTTP
Associer une demande spécifique à un pool de connexions	Regroupement de connexions
Configurer une instance de proxy	Accès à Internet via un proxy
Effectuer des demandes asynchrones pour une ressource Internet	Demandes asynchrones
Demander des données à l'aide du protocole TCP	Utilisation des services TCP
Demander des données à l'aide du protocole UDP	Utilisation des services UDP

Protocoles enfichables

Les classes **WebRequest** et **WebResponse** abstraites fournissent les classes de base des protocoles enfichables. En dérivant les classes spécifiques au protocole à partir de **WebRequest** et **WebResponse**, une application peut demander des données à une ressource Internet et lire la réponse sans spécifier le protocole utilisé.

Pour plus d'informations, consultez Programmation de protocoles enfichables.

Le tableau suivant répertorie des tâches courantes qui impliquent des protocoles enfichables.

Pour	Reportez-vous à
Dériver de **WebRequest**	Dérivation à partir de WebRequest
Dériver de **WebResponse**	Dérivation à partir de WebResponse
Propriétés spécifiques au protocole d'accès	Comment : convertir une WebRequest en propriétés spécifiques au protocole d'accès

Traçage réseau

Le traçage réseau fournit l'accès aux informations sur l'appel de méthodes et aux informations sur un trafic réseau généré par une application managée. Cette fonctionnalité permet de déboguer les applications en cours de développement et d'analyser celles qui sont déployées. La sortie fournie par un traçage réseau est personnalisable pour prendre en charge différents scénarios d'utilisation au moment du développement et dans un environnement de production.

Pour plus d'informations, consultez Traçage réseau.

Le tableau suivant répertorie des tâches courantes qui impliquent le traçage réseau.

Pour	Reportez-vous à
Activer le traçage réseau	Activation du traçage réseau
Lire les informations de traçage	Interprétation du traçage réseau
Configurer le traçage réseau	Comment : configurer le traçage réseau

Gestion du cache

Un cache fournit un stockage temporaire des ressources qui ont été demandées par une application. Si une application demande plusieurs fois la même ressource, cette dernière peut être retournée à partir du cache, ce qui évite de la demander une nouvelle fois à partir du serveur.

Une stratégie de cache définit les règles utilisées pour déterminer si une demande peut être satisfaite à l'aide d'une copie de la ressource demandée mise en cache.

Pour plus d'informations, consultez Gestion du cache pour les applications réseau.

Le tableau suivant répertorie des tâches courantes qui impliquent la gestion du cache.

Pour	Reportez-vous à
Définir une stratégie de cache basée de l'emplacement	Comment : définir une stratégie de cache reposant sur l'emplacement pour une application
Définir une stratégie de cache basée sur l'heure par défaut	Comment : définir la stratégie de cache reposant sur le temps par défaut pour une application
Personnaliser une stratégie de cache basée sur l'heure	Comment : personnaliser une stratégie de cache reposant sur le temps
Définir une stratégie de cache pour une demande	Comment : définir une stratégie de cache pour une demande

Sécurité

Les classes System.Net fournissent un environnement sécurisé pour les applications Internet en proposant une prise en charge intégrée pour les mécanismes d'authentification d'applications Internet couramment utilisés et pour les autorisations d'accès de code du .NET Framework.

Pour plus d'informations, consultez Méthodes conseillées pour les classes System.Net.

Le tableau suivant répertorie des tâches courantes qui impliquent la sécurité.

Pour	Reportez-vous à
Utiliser des connexions SSL (Secure Sockets Layer) à un serveur HTTP	Utilisation de Secure Sockets Layer
Utiliser des méthodes d'authentification HTTP pour établir une connexion authentifiée à un serveur HTTP.	Authentification Internet
Définir la sécurité d'accès du code pour les applications utilisant des connexions Internet.	Autorisations Web et Socket
Utiliser l'authentification de base et Digest.	Authentification de base et Digest
Utiliser l'authentification NTLM et Kerberos.	Authentification NTLM et Kerberos
Utiliser System.Net pour sécuriser les Services Web XML.	Sécurisation des Services Web XML créés à l'aide d'ASP.NET

Voir aussi

Référence

System.Net

System.Net.Sockets

Concepts

Méthodes conseillées pour les classes System.Net

Configuration des applications Internet

Autres ressources

Programmation réseau

Espaces de noms de réseaux dans Visual Studio

Classes utilisées dans les opérations de réseau du .NET Framework

System.Net, System.Net.Sockets et Uri fournissent une implémentation en couche des services Internet, extensible et managée, qui peut être rapidement et facilement intégrée à vos applications.

Classes de base utilisées dans les opérations de réseau

Ce tableau répertorie les classes les plus importantes utilisées pour les opérations de réseau du .NET Framework. Pour une liste plus complète, consultez la documentation du .NET Framework.

Class	Description
System.Net	Fournit une interface de programmation simple pour un grand nombre des protocoles réseau employés.
System.Net.Authorization	Contient un message d'authentification pour un serveur Internet.
Cookie	Fournit un ensemble de méthodes et de propriétés utilisées pour gérer les cookies.
WebClient	Fournit des méthodes communes pour l'envoi de données à une ressource identifiée par un URI ou pour la réception de données en provenance de cette ressource.
WebRequest	Lance une demande à un URI.
WebResponse	Fournit une réponse d'un URI.
System.Net.Sockets	Fournit une implémentation managée de l'interface Windows Sockets (Winsock) pour les développeurs qui doivent contrôler étroitement l'accès au réseau.
System.Net.Sockets.TcpClient	Fournit des connexions clientes pour les services réseau TCP.
System.Net.Sockets.TcpListener	Écoute des connexions de clients réseau TCP.
System.Net.Sockets.UdpClient	Fournit les services réseau du protocole UDP (User Datagram Protocol).
System.Uri	Fournit une représentation objet d'un URI.
UriBuilder	Fournit un constructeur et des modificateurs personnalisés pour les URI.

Voir aussi

Concepts

Concepts de base des opérations de réseau du .NET Framework

Opérations relatives au port dans le .NET Framework avec Visual Basic

Vous pouvez accéder aux ports série de votre ordinateur via les classes du .NET Framework de l'espace de noms System.IO.Ports. La classe la plus importante, SerialPort, fournit une infrastructure pour les E/S synchrones et pilotées par événements, l'accès aux états de broche et d'arrêt et l'accès aux propriétés des pilotes de série. Il peut être encapsulé dans un objet Stream, accessible via la propriété BaseStream. L'encapsulation de **SerialPort** dans un objet **Stream** permet au port série d'être accessible par les classes qui utilisent des flux. L'espace de noms inclut des énumérations qui simplifient le contrôle des ports série.

La méthode My.Computer.Ports.OpenSerialPort constitue le moyen le plus simple pour créer un objet **SerialPort**.

Remarque
Vous ne pouvez pas utiliser les classes du .NET Framework pour accéder directement aux autres de types de ports, tels que les ports parallèles, les ports USB, etc. Pour accéder au réseau de votre ordinateur, consultez Classes utilisées dans les opérations de réseau du .NET Framework.

Énumérations

Ce tableau répertorie et décrit les principales énumérations utilisées pour accéder à un port série :

Enumération	Description
Handshake	Spécifie le protocole de contrôle utilisé pour établir une communication de port série pour un objet **SerialPort**.
Parity	Spécifie le bit de parité pour un objet **SerialPort**.
SerialData	Spécifie le type de caractère qui a été reçu sur le port série de l'objet **SerialPort**.
SerialError	Spécifie les erreurs qui se produisent sur l'objet **SerialPort**
SerialPinChange	Spécifie le type de modification qui s'est produit sur l'objet **SerialPort**.
StopBits	Spécifie le nombre de bits d'arrêt utilisé sur l'objet **SerialPort**.

Tâches

Ce tableau répertorie et décrit les principales tâches utilisées pour accéder à un port série :

Pour	Reportez-vous à
Afficher les ports série disponibles	Comment : afficher les ports série disponibles dans Visual Basic
Utiliser un modem connecté au port série d'un ordinateur	Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Envoyer une chaîne au port série d'un ordinateur	Comment : envoyer des chaînes aux ports série dans Visual Basic
Recevoir une chaîne du port série d'un ordinateur	Comment : recevoir des chaînes des ports série dans Visual Basic

Voir aussi

Référence

My.Computer.Ports, objet

Autres ressources

Accès aux ports de l'ordinateur

Fonctionnalités plus poussées de Visual Basic

Sérialisation dans le .NET Framework avec Visual Basic

La sérialisation est le processus de conversion d'un objet en un flux d'octets afin de le faire persister dans la mémoire, dans une base de données ou dans un fichier. Vous pouvez enregistrer et recréer ces objets à l'aide de fonctionnalités fournies par le .NET Framework.

Dans cette section

Concepts de base de la sérialisation .NET Framework

Fournit une vue d'ensemble de la sérialisation à l'aide des classes .NET Framework.

Classes utilisées dans la sérialisation .NET Framework

Répertorie les classes .NET Framework utilisées dans la sérialisation.

Rubriques connexes

Exemples de sérialisation XML

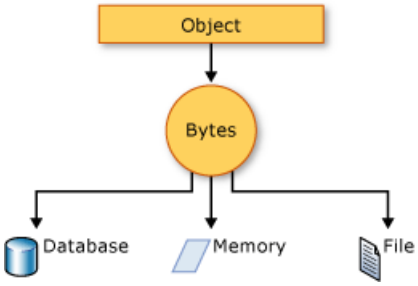
Illustre la sérialisation XML de différents objets, tels qu'un groupe de données.

Concepts de base de la sérialisation .NET Framework

La sérialisation est le processus de conversion d'un objet en un flux d'octets afin de le faire persister dans la mémoire, dans une base de données ou dans un fichier. Son objectif principal est d'enregistrer l'état d'un objet pour pouvoir le recréer si nécessaire. Le processus inverse est appelé désérialisation.

Fonctionnement de la sérialisation

Cette illustration affiche le processus global de la sérialisation.



L'objet est sérialisé à un flux qui contient non seulement les données, mais également des informations sur le type d'objet, notamment sa version, sa culture et son nom d'assembly. À partir de ce flux, il peut être stocké dans une base de données, dans un fichier ou en mémoire.

Utilisations de la sérialisation

La sérialisation permet au développeur d'enregistrer l'état d'un objet et de le recréer si nécessaire, en fournissant un stockage d'objets ainsi qu'un échange de données. Par le biais de la sérialisation, un développeur peut effectuer des actions comme l'envoi de l'objet à une application distante via un service Web, le passage d'un objet d'un domaine à un autre, via un pare-feu sous forme de chaîne XML ou la conservation des informations de sécurité ou spécifiques à l'utilisateur sur les applications.

Rendre un objet sérialisable

Pour sérialiser un objet, vous devez disposer des éléments suivants : l'objet à sérialiser, un flux pour contenir l'objet sérialisé et un Formatter. System.Runtime.Serialization contient les classes nécessaires pour la sérialisation et la désérialisation des objets.

Appliquez l'attribut SerializableAttribute à un type pour indiquer que ses instances peuvent être sérialisées. Une exception SerializationException est levée si vous essayez d'effectuer la sérialisation et que le type n'a pas l'attribut **SerializableAttribute**.

Si vous ne souhaitez pas qu'un champ de votre classe soit sérialisable, appliquez l'attribut NonSerializedAttribute. Si un champ d'un type sérialisable contient un pointeur, un handle ou une autre structure de données spécifique à un environnement particulier et ne peut pas être reconstitué de manière significative dans un autre environnement, il est possible de le rendre non sérialisable.

Si une classe sérialisée contient des références aux objets d'autres classes qui sont marquées **SerializableAttribute**, ces objets seront également sérialisés.

Sérialisation binaire et XML

La sérialisation binaire ou XML peut être utilisée. Dans une sérialisation binaire, tous les membres, même ceux qui sont en lecture seule, sont sérialisés et les performances sont améliorées. La sérialisation XML fournit un code plus lisible, ainsi qu'une plus grande souplesse du partage et de l'utilisation des objets pour des raisons d'interopérabilité.

Sérialisation binaire

La sérialisation binaire utilise le codage binaire afin de produire une sérialisation compacte destinée notamment au stockage ou au flux réseau socket. Il n'est pas convenable de faire passer les données dans un pare-feu mais les performances sont meilleures lors du stockage des données.

Sérialisation XML

La sérialisation XML sérialise les champs et les propriétés publics d'un objet, ou les paramètres et valeurs de retour des méthodes, en un flux XML conforme à un document de langage XSD (XML Schema Definition) spécifique. La sérialisation XML favorise des classes fortement typées avec des propriétés et des champs publics convertis en XML. System.Xml.Serialization contient les classes nécessaires pour la sérialisation et la désérialisation XML.

Vous pouvez appliquer les attributs à des classes et à des membres de classe pour contrôler la manière dont XmlSerializer

sérialise ou désérialise une instance de la classe. Pour plus d'informations, consultez Contrôle de la sérialisation XML à l'aide d'attributs et Attributs qui contrôlent la sérialisation XML.

Ce tableau répertorie des tâches associées à la sérialisation XML :

Pour	Reportez-vous à
Sérialiser un objet	Comment : Sérialiser un objet
Désérialiser un objet	Comment : Désérialiser un objet
Générer des classes et une documentation du schéma XML	Comment : Utiliser l'outil XML Schema Definition pour générer des classes et des documents de schéma XML
Qualifier un élément XML et des noms d'attribut XML	Comment : Qualifier des noms d'éléments et d'attributs XML
Spécifier un nom d'élément différent pour un flux XML	Comment : Spécifier un autre nom d'élément pour un flux XML
Contrôler la sérialisation des classes dérivées	Comment : Contrôler la sérialisation de classes dérivées

Sérialisation SOAP

La sérialisation XML peut également être utilisée pour sérialiser des objets en flux XML se conformant à la spécification SOAP. SOAP est un protocole basé sur XML, conçu spécifiquement pour transporter des appels de procédure à l'aide de XML. Comme avec la sérialisation XML, les attributs peuvent également être utilisés pour contrôler les messages SOAP de style littéral générés par un service Web XML. Pour plus d'informations, consultez Sérialisation XML à l'aide des services Web XML et Attributs qui contrôlent la sérialisation codée en SOAP.

Ce tableau répertorie les tâches associées à la sérialisation XML codée en SOAP :

Pour	Reportez-vous à
Sérialiser un objet en tant que flux XML codé en SOAP	Comment : Sérialiser un objet en tant que flux XML codé en SOAP
Substituer la sérialisation XML codée en SOAP	Comment : Substituer la sérialisation XML codée en SOAP

Sérialisation de base et personnalisée

La sérialisation peut être effectuée de deux manières, de base ou personnalisée. La sérialisation de base utilise le .NET Framework pour sérialiser automatiquement l'objet.

Sérialisation de base

La seule condition de la sérialisation de base est que l'attribut **SerializableAttribute** soit appliqué à l'objet. L'**NonSerializedAttribute** peut être utilisé pour empêcher que des champs spécifiques ne soient sérialisés.

Lorsque vous utilisez la sérialisation de base, le versioning des objets peut causer des problèmes, auquel cas la sérialisation personnalisée peut être préférable. La sérialisation de base constitue le moyen le plus facile pour effectuer la sérialisation, mais ne permet pas un contrôle efficace sur le processus.

Sérialisation personnalisée

Dans la sérialisation personnalisée, vous pouvez spécifier exactement les objets à sérialiser et le mode de déroulement du

processus. La classe doit être marquée **SerializableAttribute** et implémenter l'interface ISerializable.

Si vous souhaitez que votre objet soit également désérialisé de manière personnalisée, vous devez utiliser un constructeur personnalisé.

Sérialisation du concepteur

La sérialisation du concepteur est une forme de sérialisation spéciale qui implique le type de persistance des objets généralement associé aux outils de développement. La sérialisation du concepteur est le processus de conversion d'un graphique d'objets en un fichier source qui peut être utilisé ultérieurement pour récupérer le graphique d'objets. Un fichier source peut contenir du code, un balisage ou même des informations sur les tables SQL. Pour plus d'informations, consultez Vue d'ensemble de la sérialisation du concepteur.

Voir aussi

Tâches

Comment : Segmenter des données sérialisées

Concepts

Exemples de sérialisation XML

Classes utilisées dans la sérialisation .NET Framework

Classes utilisées dans la sérialisation .NET Framework

Cette rubrique répertorie les classes couramment utilisées pour la sérialisation .NET Framework. Pour consulter la documentation du .NET Framework 2.0 et rechercher une liste plus complète, consultez System.Runtime.Serialization.

Classes de base pour la sérialisation

Ce tableau répertorie et décrit les classes et espaces de noms principaux utilisés pour la sérialisation :

Classe ou espace de noms	Description
System.Runtime.Serialization	Contient les classes nécessaires pour la sérialisation et la désérialisation des objets.
System.Xml.Serialization	Contient les classes nécessaires pour la sérialisation et la désérialisation XML.
SerializableAttribute	Doit s'appliquer à un objet pour le rendre sérialisable.
NonSerializedAttribute	Peut être utilisé pour empêcher la sérialisation de champs spécifiques d'un objet.

Voir aussi

Concepts

Concepts de base de la sérialisation .NET Framework

Développement pour les Smart Devices avec Visual Basic

Visual Studio 2005 prend en charge Visual Basic pour le développement d'applications Smart Device. Il fournit les outils et l'infrastructure nécessaires pour développer des applications pour Pocket PC, Smartphone et d'autres plateformes Windows CE.

Développement d'applications pour les périphériques

D'une manière générale, vous pouvez développer deux types d'applications pour les périphériques :

- Les applications Web mobiles s'exécutent sur un serveur Web et peuvent être rendues sous différents formats sur une gamme de périphériques mobiles équipés d'un navigateur. Pour plus d'informations, consultez Création de pages Web mobiles ASP.NET.
- Les applications clientes élaborées Windows CE s'exécutent sur le périphérique lui-même. C'est cette approche que nous désignons généralement par l'expression « application pour les smart devices ».
- Lors du développement pour les smart devices, vous utilisez le même environnement Visual Studio utilisé lors du développement d'applications bureau, mais certaines différences se manifestent lorsque vous ciblez des périphériques. Ces différences sont les suivantes :
- Outils supplémentaires pour la connexion et le débogage sur un périphérique distant.
- Lorsque vous créez un projet smart device, vous devez non seulement choisir un type de projet et un modèle, mais aussi sélectionner un périphérique sur lequel l'application doit être exécutée et déboguée. Ce périphérique peut être un périphérique physique connecté à l'ordinateur de développement, un périphérique relié au réseau ou un émulateur de périphérique qui s'exécute sur l'ordinateur de développement.
- Les classes et leurs membres diffèrent lors du développement pour les périphériques. Pour déterminer si une classe et ses membres sont disponibles ou non, consultez la documentation, utilisez IntelliSense ou l'Explorateur d'objets Visual Studio lorsque le projet est actif. Pour plus d'informations, consultez .NET Compact Framework.

Création d'une application smart device

La boîte de dialogue améliorée **Nouveau projet** de Visual Studio 2005 remplace l'Assistant Application Smart Device de Visual Studio .NET 2003. Dans Visual Studio 2005, vous effectuez tous les choix concernant les types de projet et les modèles à partir de la boîte de dialogue **Nouveau projet**. Pour plus d'informations, consultez Comment : créer des applications smart device à l'aide de Visual Basic ou Visual C#.

Pour obtenir la liste de tâches relatives aux smart devices, consultez Applications Smart Device (Comment faire en Visual Basic).

Données et périphériques

Le .NET Compact Framework fournit une implémentation complète de ADO.NET pour les périphériques qui prend en charge les classes DataSet et Dataview. Cette prise en charge inclut les classes DataRelation et Constraint et d'autres classes qui définissent et manipulent le **DataSet**. Le .NET Compact Framework inclut également le fournisseur de données SQL Server .NET. Pour plus d'informations, consultez Accès aux données et prise en charge du langage XML et SQL Server Compact Edition avec le .NET Compact Framework.

Réseau

Le .NET Compact Framework fournit l'API réseau de niveau socket et les abstractions de plus haut niveau, telles que les demandes et les réponses HTTP, DNS et Web. La connectivité est assurée via des transports IrDA (Infrared Data Association) et TCP/IP à travers l'API de sockets. Pour plus d'informations, consultez Mise en réseau et connectivité.

Développement pour le Pocket PC

Ce tableau répertorie les tâches spécifiques au Pocket PC.

Pour	Reportez-vous à
Utiliser un contrôle DocumentList pour gérer les tâches de gestion de fichiers dans votre application	Comment : utiliser un contrôle DocumentList

Utiliser des éléments de formulaire dans votre application	Style de formulaire Pocket PC
Activer les applications à partir d'un bouton matériel physique avec un composant HardwareButton dans votre application	Comment : utiliser le composant HardwareButton
Utiliser un composant InputPanel dans votre application	Comment : utiliser le composant InputPanel
Sélectionner des méthodes d'entrée Pocket PC pour les entrées d'utilisateur dans votre application	Comment : définir les méthodes d'entrée Pocket PC
Envoyer et répondre à une notification dans votre application	Comment : envoyer une notification
Détecter des touches de navigation	Comment : détecter les touches de navigation

Développement pour Smartphone

Le .NET Compact Framework peut être installé sur les périphériques Smartphone Windows Mobile 2003 et les versions ultérieures.

Ce tableau répertorie les tâches spécifiques au développement Smartphone.

Pour	**Reportez-vous à**
Définir des méthodes d'entrée Smartphone dans votre application	Comment : définir les modes d'entrée Smartphone
Substituer la touche Précédent	Comment : remplacer la touche Précédent de Smartphone
Utiliser les menus Smartphone	Comment : utiliser les menus Smartphone

Voir aussi

Concepts

Rubriques Comment relatives au .NET Compact Framework

Autres ressources

Développement Pocket PC et le .NET Compact Framework
Développement de SmartPhone et le .NET Compact Framework
Développement Windows CE
Informations sur le produit .NET Compact Framework

Multithreading avancé avec Visual Basic

Les applications multithread peuvent exécuter simultanément plusieurs tâches différentes. Les rubriques de cette section décrivent comment gérer ces tâches afin qu'elles soient utilisées ensemble, avec souplesse et sans conflits.

Dans cette section

Synchronisation des threads

Décrit comment synchroniser des applications multithread.

Regroupement des threads

Décrit comment des tâches peuvent être ajoutées à une file d'attente, puis lancées lorsque de nouveaux threads sont créés.

Composants Timer thread

Décrit comment exécuter des procédures sur des threads séparés à intervalles fixes.

Techniques avancées de synchronisation

Décrit comment les applications multithread utilisent des handles d'attente et des objets Monitor pour synchroniser plusieurs threads.

Rubriques connexes

Multithreading dans Visual Basic

Introduction à l'utilisation du multithreading dans les applications Visual Basic.

Multithreading dans les composants

Fournit des liens vers des rubriques qui décrivent comment utiliser le multithreading dans la programmation de composant.

Procédure pas à pas : création d'un composant simple multithread avec Visual Basic

Montre comment créer des composants multithread.

System.Threading

Fournit les classes et les interfaces du .NET Framework qui permettent la programmation multithread.

Composants Timer thread

La classe System.Threading.Timer est utile pour exécuter périodiquement une tâche sur un thread séparé. Par exemple, vous pouvez employer un composant Timer thread pour vérifier l'état et l'intégrité d'une base de données ou pour sauvegarder des fichiers vitaux.

Exemple Timer Thread

L'exemple ci-après lance une tâche toutes les deux secondes et utilise un indicateur pour activer la méthode Dispose qui arrête la minuterie. Cet exemple publie l'état dans la fenêtre de sortie que vous devez rendre visible. Pour ce faire, appuyez sur CTRL+ALT+O avant de tester le code.

VB

```
Class StateObjClass
    ' Used to hold parameters for calls to TimerTask
    Public SomeValue As Integer
    Public TimerReference As System.Threading.Timer
    Public TimerCanceled As Boolean
End Class

Sub RunTimer()
    Dim StateObj As New StateObjClass
    StateObj.TimerCanceled = False
    StateObj.SomeValue = 1
    Dim TimerDelegate As New Threading.TimerCallback(AddressOf TimerTask)
    ' Create a timer that calls a procedure every 2 seconds.
    ' Note: There is no Start method; the timer starts running as soon as
    ' the instance is created.
    Dim TimerItem As New System.Threading.Timer(TimerDelegate, StateObj, _
                                                2000, 2000)
    StateObj.TimerReference = TimerItem  ' Save a reference for Dispose.

    While StateObj.SomeValue < 10 ' Run for ten loops.
        System.Threading.Thread.Sleep(1000)  ' Wait one second.
    End While

    StateObj.TimerCanceled = True  ' Request Dispose of the timer object.
End Sub

Sub TimerTask(ByVal StateObj As Object)
    Dim State As StateObjClass = CType(StateObj, StateObjClass)
    ' Use the interlocked class to increment the counter variable.
    System.Threading.Interlocked.Increment(State.SomeValue)
    System.Diagnostics.Debug.WriteLine("Launched new thread  " & Now)
    If State.TimerCanceled Then    ' Dispose Requested.
        State.TimerReference.Dispose()
        System.Diagnostics.Debug.WriteLine("Done  " & Now)
    End If
End Sub
```

Les composants Timer thread sont particulièrement utiles lorsque l'objet System.Windows.Forms.Timer n'est pas disponible, par exemple lorsque vous développez des applications console.

Voir aussi

Référence

SyncLock, instruction

System.Threading

Concepts

Multithreading avancé avec Visual Basic

Applications multithread

Regroupement des threads

Le regroupement des threads en pools constitue une forme de multithreading où les tâches sont ajoutées à une file d'attente et automatiquement lancées lorsque les threads sont créés. En mode de regroupement de threads, vous appelez la méthode System.Threading.ThreadPool.QueueUserWorkItem(System.Threading.WaitCallback) avec un délégué pour la procédure à exécuter et Visual Basic crée le thread et exécute votre procédure.

Exemple de Pooling de threads

L'exemple ci-après montre comment utiliser le regroupement de threads pour lancer plusieurs tâches.

VB

```
Sub DoWork()
    ' Queue a task
    System.Threading.ThreadPool.QueueUserWorkItem( _
        New System.Threading.WaitCallback(AddressOf SomeLongTask))
    ' Queue another task
    System.Threading.ThreadPool.QueueUserWorkItem( _
        New System.Threading.WaitCallback(AddressOf AnotherLongTask))
End Sub
Sub SomeLongTask(ByVal state As Object)
    ' Insert code to perform a long task.
End Sub
Sub AnotherLongTask(ByVal state As Object)
    ' Insert code to perform another long task.
End Sub
```

Le regroupement de threads est utile pour lancer de nombreuses tâches distinctes sans définir individuellement les propriétés de chaque thread. Chaque thread démarre avec une taille de pile et une priorité par défaut. Par défaut, jusqu'à 25 threads ThreadPool peuvent s'exécuter par processeur système. Des threads supplémentaires au-delà de la limite définie peuvent être mis en file d'attente, mais ils ne démarrent que lorsque les autres threads ont terminé.

L'un des avantages du regroupement de threads est que vous pouvez transmettre des arguments d'un objet état à la procédure de la tâche. Si la procédure que vous appelez exige plusieurs arguments, vous pouvez transtyper (cast) une structure ou une instance de classe en type de données **Object**.

Paramètres des pools de thread et valeurs de retour

Le renvoi de valeurs à partir d'un thread ThreadPool n'est pas simple. La méthode standard pour renvoyer des valeurs à partir d'un appel de fonction n'est pas admise, car les procédures **Sub** constituent le seul type de procédure à pouvoir être mis en file d'attente vers un pool de threads. L'une des méthodes de transmission des paramètres et des valeurs de retour consiste à encapsuler les paramètres, les valeurs de retour et les méthodes dans une classe wrapper, comme indiqué dans Paramètres et valeurs de retour pour les procédures multithread.

Une méthode plus simple consiste à fournir des paramètres et des valeurs de retour à l'aide de la variable objet état **ByVal** (facultative) de la méthode **QueueUserWorkItem**. Si vous utilisez cette variable pour transmettre une référence à une instance de classe, les membres de l'instance peuvent être modifiés par le thread ThreadPool et utilisés comme valeurs de retour.

À première vue, il ne vous semble peut-être pas évident de pouvoir modifier un objet référencé par une variable transmise par valeur. Il est possible de le faire, car seule la référence d'objet est transmise par valeur. Lorsque vous apportez des modifications aux membres de l'objet désigné par la référence d'objet, celles-ci s'appliquent à l'instance de la classe réelle.

Les structures ne peuvent pas être employées pour renvoyer des valeurs dans les objets état. Étant donné que les structures représentent des types valeur, les modifications que le processus asynchrone effectue n'ont aucune incidence sur les membres de la structure d'origine. Utilisez des structures pour fournir des paramètres lorsque les valeurs de retour ne sont pas nécessaires.

Voir aussi

Référence

SyncLock, instruction
QueueUserWorkItem

System.Threading

Concepts

Multithreading avancé avec Visual Basic

Applications multithread

Synchronisation des threads

Multithreading avec des formulaires et contrôles

Synchronisation des threads

Lorsque vous écrivez une application à threads, vous pouvez être amené à synchroniser différents threads séparés avec d'autres parties de votre programme. La synchronisation apporte un compromis entre la nature non structurée de la programmation multithread et l'ordre structuré du traitement synchrone.

Vous utilisez les techniques de synchronisation pour :

- contrôler explicitement l'ordre d'exécution du code dès que des tâches doivent être effectuées dans une séquence déterminée ;

  - ou -

- pour empêcher les problèmes qui peuvent se produire lorsque deux threads partagent simultanément une même ressource.

Par exemple, vous pouvez utiliser la synchronisation pour forcer une procédure d'affichage à attendre la fin d'une procédure de récupération de données qui s'exécute sur un autre thread.

Techniques de synchronisation

Il existe deux approches en matière de synchronisation, l'interrogation et la manipulation d'objets de synchronisation. Pour plus d'informations sur les objets de synchronisation, consultez Techniques avancées de synchronisation.

Interrogation

L'interrogation répétée permet de vérifier l'état d'un appel asynchrone à partir d'une boucle. L'interrogation est la méthode la moins efficace pour gérer les threads : la vérification répétée de l'état des diverses propriétés de thread entraîne un gaspillage des ressources.

Par exemple, vous pouvez lancer une interrogation à l'aide de la propriété IsAlive pour savoir si un thread est fermé. Utilisez cette propriété avec précaution, car un thread actif n'est pas nécessairement en cours d'exécution.

Vous pouvez obtenir de plus amples informations sur l'état d'un thread à l'aide de la propriété ThreadState de ce thread. Comme les threads peuvent être dans plusieurs états à un même moment, la valeur stockée dans **ThreadState** peut être une combinaison des valeurs de l'énumération System.Threading.ThreadState. Par conséquent, vous devez vérifier attentivement tous les états pertinents d'un thread lors de l'interrogation. Par exemple, si un thread n'est pas à l'état Running, cela signifie qu'il est peut-être terminé. Toutefois, il peut également être suspendu ou mis en veille.

Attente de la fin d'un thread

La méthode System.Threading.Thread.Join permet de déterminer si un thread a pris fin avant de démarrer une autre tâche. La méthode **Join** attend un laps de temps spécifié que le thread prenne fin. Si le thread prend fin dans le délai imparti, **Join** retourne **True**; sinon, il retourne la valeur **False**.

Comme vous pouvez l'imaginer, l'interrogation sacrifie de nombreux avantages offerts par le mode multithread au bénéfice du contrôle exercé sur l'ordre d'exécution des threads. L'interrogation est si inefficace qu'elle n'est généralement pas recommandée. Une meilleure approche consiste à utiliser la méthode **Join** pour contrôler les threads. **Join** force une procédure appelante à attendre la fin de l'exécution d'un thread ou le dépassement de délai de l'appel si un délai d'attente est précisé. Le nom de cette méthode, « Join », est basé sur l'idée que la création d'un thread représente une branche dans le chemin d'exécution. Vous utilisez **Join** pour fusionner des chemins d'exécution distincts en un seul thread à nouveau.

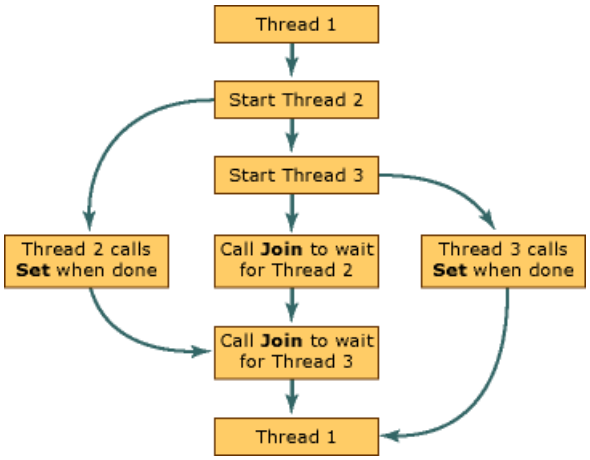


Figure 1 Modèle de thread

Important : notez que **Join** correspond à un appel synchrone ou *bloquant*. Lorsque vous appelez **Join** ou une méthode d'attente d'un handle d'attente, la procédure appelante s'arrête et attend que le thread signale que l'opération est terminée.

VB

```
Sub JoinThreads()
    Dim Thread1 As New System.Threading.Thread(AddressOf SomeTask)
    Thread1.Start()
    Thread1.Join()      ' Wait for the thread to finish.
    MsgBox("Thread is done")
End Sub
Sub SomeTask()
    ' Insert code to perform a task here.
End Sub
```

Ces méthodes simples de contrôle des threads, qui sont utiles lorsque vous gérez un petit nombre de threads, sont difficiles à employer avec les projets importants. Techniques avancées de synchronisation aborde certaines techniques que vous pouvez utiliser pour synchroniser les threads

Voir aussi

Concepts

Multithreading avancé avec Visual Basic

Paramètres et valeurs de retour pour les procédures multithread

Autres ressources

Multithreading dans Visual Basic

Techniques avancées de synchronisation

Les applications multithread utilisent souvent des handles d'attente et des objets Monitor pour synchroniser plusieurs threads. Ces sections expliquent comment utiliser les classes du .NET Framework suivantes lors de la synchronisation des threads : AutoResetEvent, Interlocked, ManualResetEvent, Monitor, Mutex, ReaderWriterLock, Timer, WaitHandle.

Handles d'attente

Les handles d'attente sont des objets qui signalent l'état d'un thread à un autre thread. Les threads peuvent utiliser des handles d'attente pour notifier à d'autres threads qu'ils ont besoin d'un accès exclusif à une ressource. Les autres threads doivent alors attendre que le handle d'attente ne soit plus en vigueur pour pouvoir utiliser la ressource réservée. Les handles d'attente peuvent prendre deux états, *signaled* (signalé) et *nonsignaled* (non signalé). Un handle d'attente qui n'appartient à aucun thread est à l'état *signalé*. Un handle d'attente qui appartient à un thread est à l'état *non signalé*.

Les threads demandent la propriété d'un handle d'attente en appelant une des méthodes d'attente, comme WaitOne, WaitAny ou WaitAll. Ces méthodes d'attente bloquent les appels qui ressemblent à la méthode Join d'un thread individuel :

- Si aucun autre thread ne possède le handle d'attente, l'appel retourne immédiatement la valeur **True**, le handle d'attente passe à l'état non signalé et le thread propriétaire du handle d'attente poursuit son exécution.
- Si un thread appelle l'une des méthodes d'attente d'un handle d'attente, mais que ce handle d'attente est détenu par un autre thread, le thread appelant attend pendant un certain délai (si un délai d'attente est indiqué) ou indéfiniment (si aucun délai d'attente n'est indiqué) que l'autre thread libère le handle d'attente. Si un délai d'attente est indiqué et que le handle d'attente est libéré avant la fin de celui-ci, l'appel retourne la valeur **True**. Sinon, il retourne la valeur **False** et le thread appelant poursuit son exécution.

Les threads propriétaires d'un handle d'attente appellent la méthode Set lorsqu'ils ont terminé ou qu'ils n'ont plus besoin du handle d'attente. Les autres threads peuvent faire repasser le handle d'attente à l'état non signalé. Pour ce faire, soit ils appellent la méthode Reset, soit ils appellent **WaitOne**, **WaitAny** ou **WaitAll** et attendent avec succès qu'un thread appelle **Set**. Les handles **AutoResetEvent** sont automatiquement réinitialisés par le système après la libération d'un seul thread en attente Si aucun thread n'est en attente, l'objet événement reste à l'état signalé.

Trois types de handles d'attente sont couramment utilisés avec Visual Basic : les objets mutex, **ManualResetEvent** et **AutoResetEvent**. Les deux derniers handles d'attente sont souvent désignés « événements de synchronisation ».

Objets mutex

Les objets mutex sont des objets de synchronisation qui ne peuvent appartenir qu'à un seul thread à la fois. La propriété des objets mutex est mutuellement exclusive, d'où le nom « mutex ». Les threads demandent la propriété d'un objet mutex lorsqu'ils ont besoin d'un accès exclusif à une ressource. Étant donné qu'un seul thread à la fois peut être propriétaire d'un objet mutex, les autres threads doivent attendre la propriété de cet objet mutex pour pouvoir utiliser la ressource voulue.

La méthode **WaitOne** force un thread appelant à attendre la propriété d'un objet mutex. Si un thread propriétaire d'un objet mutex se termine normalement, l'objet mutex passe à l'état signalé et le thread en attente suivant obtient la propriété de l'objet.

Événements de synchronisation

Les événements de synchronisation notifient aux autres threads qu'un événement s'est produit ou qu'une ressource est disponible. Même si l'expression comporte le terme « événement », les événements de synchronisation sont différents des autres événements Visual Basic ; ils constituent vraiment des handles d'attente. Comme les autres handles d'attente, les événements de synchronisation peuvent prendre deux états, signalé et non signalé.

Les threads qui appellent l'une des méthodes d'attente d'un événement de synchronisation doivent attendre qu'un autre thread signale l'événement par l'appel de la méthode **Set**. Il existe deux classes d'événements de synchronisation : **ManualResetEvent** et **AutoResetEvent**.

Les threads font passer les instances de **ManualResetEvent** à l'état signalé par la méthode **Set**. Ils font passer les instances de **ManualResetEvent** à l'état non signalé par la méthode **Reset** ou lorsque le contrôle revient à un appel **WaitOne** en attente.

Les instances de la classe **AutoResetEvent** peuvent également être signalées à l'aide de **Set**, mais elles reviennent automatiquement à l'état non signalé dès qu'un thread en attente est informé que l'événement est désormais signalé.

Objets Monitor et SyncLock

Les objets Monitor permettent de s'assurer qu'un bloc de code s'exécute sans être interrompu par du code s'exécutant sur d'autres threads. En d'autres termes, le code des autres threads ne peut s'exécuter que si le code du bloc synchronisé a terminé.

Par exemple, soit un programme qui lit des données de manière répétée et asynchrone, puis affiche les résultats. Avec les systèmes d'exploitation qui utilisent le mode multitâche préemptif, un thread en cours d'exécution peut être interrompu par le système d'exploitation qui accorde un délai pour qu'un autre thread puisse s'exécuter. Sans la synchronisation, vous pouvez obtenir une vue des données partiellement mise à jour si l'objet qui représente ces données est modifié par un autre thread alors que les données sont en cours d'affichage. Les objets Monitor garantissent l'exécution ininterrompue d'une section de code. Visual Basic fournit les instructions **SyncLock** et **End SyncLock** pour simplifier l'accès aux objets Monitor. Visual C# utilise le mot clé **Lock** de la même façon.

Remarque
L'accès à un objet n'est verrouillé que si le code d'accès est contenu dans un verrou **SyncLock** sur la même instance d'objet.

Pour plus d'informations sur l'instruction **SyncLock**, consultez SyncLock, instruction.

Classe Interlocked

Vous pouvez utiliser les méthodes de la classe **Interlocked** pour éviter les problèmes qui peuvent se produire lorsque plusieurs threads tentent simultanément de mettre à jour ou de comparer une même valeur. Les méthodes de cette classe vous permettent d'incrémenter, de décrémenter, d'échanger et de comparer des valeurs d'un thread en toute sécurité. L'exemple ci-après montre comment utiliser la méthode Increment pour incrémenter une variable qui est partagée par des procédures s'exécutant sur des threads séparés.

VB

```
Sub ThreadA(ByRef IntA As Integer)
    System.Threading.Interlocked.Increment(IntA)
End Sub

Sub ThreadB(ByRef IntA As Integer)
    System.Threading.Interlocked.Increment(IntA)
End Sub
```

Verrous ReaderWriter

Dans certains cas, vous pouvez souhaiter verrouiller une ressource uniquement lorsque les données sont en cours d'écriture et autoriser plusieurs clients à lire simultanément des données qui ne sont pas en cours de mises à jour. La classe **ReaderWriterLock** met en œuvre l'accès exclusif à une ressource en cours de modification par un thread, mais elle permet d'accéder en mode non exclusif à cette ressource pour la lire. Les verrous ReaderWriter constituent une alternative utile aux verrous exclusifs qui provoquent l'attente des autres threads, même lorsque ces threads n'ont pas besoin de mettre à jour des données.

Interblocages

La synchronisation des threads est d'une aide inestimable dans les applications multithread, mais le risque de création d'un *deadlock* subsiste lorsque plusieurs threads s'attendent mutuellement et que l'application s'arrête. On peut comparer l'interblocage au cas d'automobiles bloquées à un carrefour où chaque conducteur attend que l'autre passe. Il est important d'éviter les interblocages, la solution étant une planification soigneuse. Vous pouvez souvent prévoir les situations d'interblocage à l'aide d'un schéma des applications multithread élaboré avant le démarrage du codage.

Voir aussi

Référence

SyncLock, instruction
System.Threading

Concepts

Multithreading avancé avec Visual Basic
Synchronisation des threads
États d'un thread
Applications multithread

Autres ressources

Multithreading dans les composants

Concepts d'applications Windows Forms en Visual Basic

Les applications Windows Forms sont des applications clientes qui s'exécutent sur l'ordinateur de l'utilisateur, affichent des informations, demandent des entrées de données aux utilisateurs et communiquent avec des ordinateurs distants sur un réseau. Lorsque vous développez des applications de plus grande envergure, vous pouvez être amené à explorer le .NET Framework et les classes qu'il propose. Cette section fournit une vue d'ensemble des classes et des concepts du .NET Framework relatifs aux applications Windows Forms.

Dans cette section

Concepts de base de l'application Windows Forms

Fournit une vue d'ensemble de la création d'applications Windows Forms à l'aide des classes .NET Framework.

Classes utilisées dans les formulaires .NET Framework

Répertorie les classes .NET Framework utilisées dans les applications Windows Forms.

Rubriques connexes

Vue d'ensemble des applications Windows

Aborde les détails relatifs à la création d'applications Windows Forms à l'aide de Visual Studio 2005.

Vue d'ensemble des Windows Forms

Décrit les avantages des applications clientes intelligentes et les principales fonctionnalités de programmation Windows Forms.

Voir aussi

Référence

Vue d'ensemble des Windows Forms

Concepts

Vue d'ensemble du modèle d'application Visual Basic

Autres ressources

Fonctionnalités plus poussées de Visual Basic

Concepts de base de l'application Windows Forms

La possibilité de créer des applications Windows Forms qui s'exécutent localement sur les ordinateurs des utilisateurs constitue un élément essentiel de Visual Basic. Vous pouvez utiliser Visual Studio 2005 pour créer l'application et l'interface utilisateur à l'aide de Windows Forms. Une application Windows Forms repose sur les classes de l'espace de noms System.Windows.Forms.

Conception d'applications Windows Forms

Vous pouvez créer des applications de service Windows Forms et Windows à l'aide de Visual Studio. Pour plus d'informations, consultez les rubriques suivantes :

- Mise en route des Windows Forms. Fournit des informations sur la manière de créer et de programmer des Windows Forms.
- Rubriques Comment et Procédure pas à pas relatives aux Windows Forms. Répertorie les rubriques qui permettent de développer pas à pas les applications Windows Forms les plus fréquemment créées, basées sur les Windows Forms.
- Contrôles Windows Forms. Collection de rubriques expliquant en détail l'utilisation des contrôles Windows Forms.
- Applications de service Windows. Répertorie les rubriques qui expliquent comment créer des services Windows.

Génération d'interfaces utilisateur complètes et interactives

Windows Forms est le composant client intelligent du .NET Framework, un ensemble de bibliothèques managées qui permettent des tâches d'application courantes telles que la lecture et l'écriture dans le système de fichiers. À l'aide d'un environnement de développement comme Visual Studio, vous pouvez créer des applications Windows Forms qui affichent des informations, demandent des entrées de données aux utilisateurs et communiquent avec les ordinateurs distants sur un réseau.

Dans Windows Forms, un formulaire est une surface visuelle sur laquelle vous affichez des informations à l'intention de l'utilisateur. Généralement, vous créez des applications Windows Forms en plaçant des contrôles sur les formulaires et en développant des réponses aux actions des utilisateurs telles les clics de souris ou les pressions sur les touches. Un *contrôle* est un élément d'interface utilisateur discret qui affiche des données ou accepte une entrée de données.

Événements

Lorsqu'un utilisateur effectue une action sur votre formulaire ou sur un de ses contrôles, il génère un événement. Votre application réagit à ces événements en utilisant du code et traite les événements lorsqu'ils se produisent. Pour plus d'informations, consultez Création de gestionnaires d'événements dans les Windows Forms.

Contrôles

Windows Forms contient de nombreux contrôles que vous pouvez placer sur les formulaires : contrôles qui affichent des zones de texte, des boutons, zones déroulantes, des cases d'option et même des pages Web. Pour obtenir la liste de tous les contrôles que vous pouvez utiliser sur un formulaire, consultez Contrôles à utiliser dans les Windows Forms. Si un contrôle existant ne satisfait pas vos besoins, Windows Forms prend également en charge la création de vos propres contrôles personnalisés à l'aide de la classe UserControl.

Windows Forms possède des contrôles d'interface utilisateur complets qui émulent des fonctions dans des applications haut de gamme comme Microsoft Office. À l'aide du contrôle ToolStrip et MenuStrip, vous pouvez créer des barres d'outils et des menus qui contiennent du texte et des images, afficher des sous-menus et héberger d'autres contrôles tels que des zones de texte et des zones de liste déroulante.

À l'aide du Concepteur de formulaires de glisser-déplacer Visual Studio, vous pouvez facilement créer des applications Windows Forms : sélectionnez simplement les contrôles avec votre curseur et placez-les sur le formulaire à l'emplacement de votre choix. Le concepteur fournit des outils tels que le quadrillage et des « lignes d'alignement » pour résoudre le problème d'alignement des contrôles. Lorsque vous utilisez Visual Studio ou effectuez une compilation sur la ligne de commande, vous pouvez utiliser les contrôles FlowLayoutPanel, TableLayoutPanel et SplitContainer pour créer des dispositions de formulaire avancées avec un minimum de temps et d'effort.

Éléments d'interface personnalisés

Enfin, si vous devez créer vos propres éléments d'interface personnalisés, l'espace de noms System.Drawing contient toutes les classes nécessaires pour rendre des lignes, des cercles et d'autres formes directement sur un formulaire.

Pour des informations pas à pas sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes.

Pour	Reportez-vous à
Créer une application Windows Forms dans Visual Studio.	Procédure pas à pas : création d'un Windows Form simple
Utiliser des contrôles sur les formulaires	Comment : ajouter des contrôles à des Windows Forms
Gérer des événements à partir d'un formulaire et de ses contrôles	Comment : créer des gestionnaires d'événements à l'aide du concepteur
Utilisez le contrôle **ToolStrip**	Comment : créer un ToolStrip Windows Forms de base avec des éléments standard à l'aide du concepteur
Créer des graphiques à l'aide de **System.Drawing**	Mise en route de la programmation graphique
Créer des contrôles personnalisés	Comment : hériter de la classe UserControl

Affichage et manipulation de données

De nombreuses applications doivent afficher les données dans une base de données, un fichier XML, un service Web XML ou une autre source de données. Windows Forms fournit un contrôle souple appelé contrôle DataGridView pour rendre ces données tabulaires dans un format de ligne et de colonne traditionnel de sorte que chaque donnée occupe sa propre cellule. À l'aide de **DataGridView**, vous pouvez personnaliser l'apparence des cellules, verrouiller les lignes et les colonnes arbitraires en place et afficher des contrôles complexes dans les cellules, entre autres fonctionnalités.

La connexion aux sources de données sur un réseau est une tâche simple dans les clients intelligents Windows Forms. Le composant BindingSource, nouveau avec Windows Forms dans Visual Studio 2005 et le .NET Framework 2.0, représente une connexion à une source de données et expose des méthodes de liaison des données aux contrôles, de navigation vers les enregistrements précédents et suivants, de modification des enregistrements et d'enregistrement des modifications dans la source d'origine. Le contrôle BindingNavigator fournit une interface simple sur le composant **BindingSource** afin que les utilisateurs puissent naviguer entre les enregistrements.

Contrôles liés aux données

Vous pouvez créer facilement des contrôles liés aux données à l'aide de la fenêtre Sources de données qui affiche des sources de données telles que des bases de données, des services Web et des objets dans votre projet. Vous pouvez créer des contrôles liés aux données en faisant glisser des éléments depuis cette fenêtre jusqu'à votre projet. Vous pouvez également connecter des contrôles existants à des données en faisant glisser des objets depuis la fenêtre Sources de données jusqu'à des contrôles existants.

Valeurs

Les paramètres constituent un autre type de liaison de données que vous pouvez gérer dans Windows Forms. La plupart des applications clientes intelligentes doivent conserver des informations sur leur état à l'exécution, telles que la dernière taille des formulaires, et conserver des données sur les préférences de l'utilisateur, telles que les emplacements par défaut des fichiers enregistrés. La fonction des paramètres d'application prend en charge ces conditions en fournissant un moyen facile de stocker les deux types de paramètres sur l'ordinateur client. Une fois définis à l'aide de Visual Studio ou d'un éditeur de code, ces paramètres sont rendus persistants au format XML et automatiquement reconvertis dans la mémoire au moment de l'exécution.

Pour des informations pas à pas sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes.

Pour	Reportez-vous à
Utiliser un composant **BindingSource**	Comment : lier des contrôles Windows Forms au composant BindingSource à l'aide du concepteur

Utiliser les sources de données ADO.NET	Comment : trier et filtrer des données ADO.NET avec le composant BindingSource Windows Forms
Ouvrir la fenêtre Sources de données	Procédure pas à pas : affichage de données sur un formulaire dans une application Windows
Utiliser les paramètres d'application	Comment : créer des paramètres d'application à l'aide du concepteur

Déploiement d'applications sur les ordinateurs clients

Une fois que vous avez écrit votre application, vous devez l'envoyer à vos utilisateurs afin qu'ils puissent l'installer et l'exécuter sur leurs propres ordinateurs clients. À l'aide de la technologie ClickOnce, vous pouvez déployer vos applications dans Visual Studio en utilisant simplement quelques clics et en fournissant une URL pointant vers votre application sur le Web. ClickOnce gère tous les éléments et dépendances de votre application et vérifie que l'application est correctement installée sur l'ordinateur client.

Les applications ClickOnce peuvent être configurées pour s'exécuter uniquement lorsque l'utilisateur est connecté au réseau ou pour s'exécuter à la fois en ligne et hors connexion. Lorsque vous spécifiez qu'une application doit prendre en charge une opération hors connexion, ClickOnce ajoute un lien à votre application dans le menu **Démarrer** de l'utilisateur afin que l'utilisateur puisse l'ouvrir sans utiliser l'URL.

Lorsque vous mettez à jour votre application, vous publiez un nouveau manifeste de déploiement et une nouvelle copie de votre application sur votre serveur Web. ClickOnce détecte qu'une mise à jour est disponible et met à niveau l'installation de l'utilisateur. Aucune programmation personnalisée n'est requise pour mettre à jour des assemblys anciens.

Pour une présentation complète de ClickOnce, consultez Vue d'ensemble du déploiement ClickOnce. Pour des informations pas à pas sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes :

Pour	**Reportez-vous à**
Déployer une application à l'aide de ClickOnce	Comment : publier une application ClickOnce Procédure pas à pas : déploiement manuel d'une application ClickOnce
Mettre à jour un déploiement ClickOnce	Comment : gérer des mises à jour pour une application ClickOnce
Rechercher les mises à jour de l'application	Comment : rechercher les mises à jour de l'application ClickOnce
Gérer la sécurité à l'aide de ClickOnce	Comment : activer les paramètres de sécurité ClickOnce

Autres contrôles et fonctionnalités

Il existe de nombreuses autres fonctionnalités dans Windows Forms qui rendent l'implémentation des tâches courantes rapide et facile, par exemple la prise en charge de la création de boîtes de dialogue, de l'impression, de l'ajout d'une aide et d'une documentation et de la traduction de votre application en plusieurs langues. En outre, Windows Forms compte sur le système de sécurité fiable du .NET Framework, ce qui vous permet de publier des applications plus sécurisées pour vos clients.

Pour des informations pas à pas sur l'utilisation de ces fonctionnalités, consultez les rubriques d'aide suivantes :

Pour	**Reportez-vous à**
Imprimer le contenu d'un formulaire	Comment : imprimer des graphiques dans Windows Forms Comment : imprimer un fichier texte composé de plusieurs pages dans les Windows Forms
Globaliser une application Windows Forms	Procédure pas à pas : localisation de Windows Forms
En savoir plus sur la sécurité Windows Forms	Vue d'ensemble de la sécurité dans les Windows Forms

Voir aussi

Référence

Vue d'ensemble des Windows Forms

My.Forms, objet

My.Application, objet

Concepts

Vue d'ensemble des applications Windows

Classes utilisées dans les formulaires .NET Framework

Classes utilisées dans les formulaires .NET Framework

Les tableaux suivants répertorient les classes couramment utilisées pour les applications Windows Forms du .NET Framework, regroupées en catégories dans les classes de base, les classes de contrôle, les classes de composant et les classes de boîte de dialogue.

Pour consulter la documentation du .NET Framework et rechercher une liste plus complète, consultez l'espace de noms System.Windows.Forms.

Les classes de base

Le tableau suivant répertorie et décrit les principales classes Windows Forms dans l'espace de noms **System.Windows.Forms**.

Classe	Description
Control	Définit la classe de base pour les contrôles, qui sont des composants dotés d'une représentation visuelle. Fournit les fonctionnalités de base pour tous les contrôles affichés dans un **Form**.
Form	Représente une fenêtre normale, une fenêtre non modale, une boîte de dialogue, un client de l'interface multidocument (MDI, Multiple Document Interface) ou une fenêtre parente MDI. Les formulaires composent l'interface utilisateur d'une application.
UserControl	Fournit un contrôle vide qui peut être utilisé pour créer d'autres contrôles ou créer un contrôle personnalisé qui est un composite d'autres contrôles.

Classes de contrôle

Le tableau suivant répertorie et décrit un échantillonnage des classes de contrôle de l'espace de noms **System.Windows.Forms**. Vous pouvez utiliser ces contrôles pour créer des interfaces utilisateur complètes.

Classe	Description
TextBox	Contrôle de saisie de données. Représente un contrôle de zone de texte Windows.
ComboBox	Contrôle de saisie de données. Représente un contrôle de zone de liste déroulante Windows.
Label	Contrôle d'affichage de données. Représente une étiquette Windows standard.
ListView	Contrôle d'affichage de données. Représente un contrôle list-view Windows qui affiche une collection d'éléments pouvant être affichés sous l'une des quatre différentes vues.
Button	Contrôle bouton de commande. Représente un contrôle bouton Windows.
ToolStrip	Contrôle bouton de commande. Fournit un conteneur pour les objets de barre d'outils Windows.
PropertyGrid	Contrôle de concepteur. Fournit une interface utilisateur pour parcourir les propriétés d'un objet.

Classes de composant

Outre les contrôles, l'espace de noms **System.Windows.Forms** fournit d'autres classes qui, bien qu'elles ne dérivent pas de la classe **Control**, fournissent des fonctionnalités visuelles à une application Windows. Le tableau suivant affiche quelques-unes des classes de composant disponibles.

Classe	Description
ToolTip	Composant d'affichage de données. Représente une petite fenêtre contextuelle rectangulaire qui affiche une brève description de l'objectif d'un contrôle lorsque l'utilisateur positionne le pointeur sur ce dernier.

ErrorProvider	Composant d'affichage de données. Fournit une interface utilisateur pour indiquer qu'une erreur est associée à un contrôle de formulaire.
ToolStripDropDownMenu	Composant de menu. Fournit les fonctionnalités de base du contrôle ContextMenuStrip.
ContextMenuStrip	Composant de menu. Représente un menu contextuel.
Help	Composant d'aide. Encapsule le moteur HTML Help 1.0.
HelpProvider	Composant d'aide. Fournit une aide contextuelle ou en ligne pour les contrôles.

Classes de boîte de dialogue

Windows propose un certain nombre de boîtes de dialogue communes pouvant être utilisées pour conférer à votre application une interface utilisateur cohérente lors de l'exécution de tâches telles que l'ouverture et l'enregistrement de fichiers, la manipulation des polices ou des couleurs du texte, ou d'une impression. Le tableau suivant répertorie quelques-unes des classes couramment utilisées pour lire et écrire dans les fichiers à l'aide de flux.

Classe	Description
OpenFileDialog	Fournit des fonctionnalités pour afficher une boîte de dialogue qui permet à l'utilisateur de parcourir et d'entrer le nom d'un fichier à ouvrir ou à enregistrer.
SaveFileDialog	Fournit des fonctionnalités pour afficher une boîte de dialogue qui permet à l'utilisateur de parcourir et d'entrer le nom d'un fichier à ouvrir ou à enregistrer.
FontDialog	Affiche une boîte de dialogue pour modifier les éléments de la police utilisée par votre application
PageSetupDialog	Affiche une boîte de dialogue qui permet à l'utilisateur de contrôler les aspects de l'impression de documents.
PrintDialog	Affiche une boîte de dialogue qui permet à l'utilisateur de contrôler les aspects de l'impression de documents.
MessageBox	Affiche un message qui peut afficher et récupérer des données de l'utilisateur.

Voir aussi

Référence

Vue d'ensemble des Windows Forms
System.Windows.Forms
System.Drawing.Printing

Concepts

Concepts d'applications Windows Forms en Visual Basic
Concepts de base de l'application Windows Forms
Vue d'ensemble des applications Windows

Ajout de rapports affichables aux applications Visual Studio

Visual Studio prend en charge diverses solutions de création de rapport pour vous aider à ajouter des rapports de données élaborés à vos applications Visual Basic. Vous pouvez créer et ajouter des rapports à l'aide de contrôles ReportViewer, de Crystal Reports ou de SQL Server Reporting Services.

Remarque
SQL Server Reporting Services fait partie de SQL Server 2005 plutôt que de Visual Studio 2005. Reporting Services n'est pas installé sur votre système à moins que vous n'ayez installé SQL Server 2005.

Cette page décrit les contrôles ReportViewer et Reporting Services. Pour plus d'informations sur Crystal Reports, consultez Vue d'ensemble de Crystal Reports.

Vue d'ensemble de la technologie de création de rapports de Microsoft dans les applications Visual Basic

Choisissez l'une des approches suivantes pour utiliser une technologie de création de rapports Microsoft dans votre application :

- Ajouter une ou plusieurs instances d'un contrôle ReportViewer à une application Windows Visual Basic.
- Intégrer SQL Server Reporting Services par programme en appelant le service Web Report Server.
- Utiliser conjointement le contrôle ReportViewer et Microsoft SQL Server 2005 Reporting Services, en utilisant le contrôle comme une visionneuse de rapports et un serveur de rapports comme processeur de rapport. (Notez que vous devez utiliser la version SQL Server 2005 de Reporting Services si vous souhaitez utiliser conjointement un serveur de rapports et le contrôle ReportViewer).

Utilisation de contrôles ReportViewer

La façon la plus simple d'incorporer les fonctionnalités de rapport dans une application Windows Visual Basic est d'ajouter le contrôle ReportViewer à un formulaire dans votre application. Le contrôle ajoute directement des fonctions de traitement de rapports à votre application et fournit un Concepteur de rapports intégré afin que vous puissiez générer des rapports à l'aide de données provenant de n'importe quel objet de données ADO.NET. Une API complète fournit l'accès par programme au contrôle et aux rapports afin que vous puissiez configurer les fonctionnalités à l'exécution.

ReportViewer fournit le traitement intégré de rapports et une fonction d'affichage dans un contrôle de données unique et librement distribuable. Choisissez des contrôles ReportViewer si vous avez besoin des fonctionnalités de rapport suivantes :

- Traitement des rapports dans l'application cliente. Un rapport traité apparaît dans une zone d'affichage fournie par le contrôle.
- Liaison des données à des tables de données ADO.NET. Vous pouvez créer des rapports qui utilisent des instances de DataTable fournies au contrôle. Vous pouvez également créer directement une liaison à des objets métier.
- Contrôles redistribuables que vous pouvez inclure dans votre application.
- Fonctionnalités au moment de l'exécution, telles que la navigation entre les pages, l'impression, la recherche et les formats d'exportation. Une barre d'outils ReportViewer fournit la prise en charge de ces opérations.

Pour utiliser le contrôle ReportViewer, vous pouvez le faire glisser de la section **Données** de la Boîte à outils Visual Studio vers un formulaire dans votre application Windows Visual Basic. Pour en savoir plus sur ReportViewer, consultez ReportViewer Controls (Visual Studio) et Samples and Walkthroughs.

Création de rapports dans Visual Studio pour des contrôles ReportViewer

Pour générer un rapport qui s'exécute dans ReportViewer, ajoutez un modèle de **Rapport** à votre projet. Visual Studio crée un fichier de définition de rapport client (.rdlc), ajoute le fichier à votre projet et ouvre un Concepteur de rapports intégré dans l'espace de travail Visual Studio.

Le Concepteur de rapports Visual Studio s'intègre dans la fenêtre **Sources de données**. Lorsque vous faites glisser un champ de la fenêtre **Sources de données** vers le rapport, le Concepteur de rapports copie les métadonnées de la source de données dans le fichier de définition du rapport. Ces métadonnées sont utilisées par le contrôle ReportViewer pour générer automatiquement le code de liaison de données.

Le Concepteur de rapports Visual Studio n'inclut pas de fonctionnalité d'aperçu du rapport. Pour afficher un aperçu de votre rapport, exécutez l'application et affichez l'aperçu du rapport qu'elle contient.

Pour ajouter des fonctionnalités de rapport de base à votre application

1. Faites glisser un contrôle ReportViewer de l'onglet **Données** de la **Boîte à outils** sur votre formulaire.
2. Dans le menu **Projet**, choisissez **Ajouter un nouvel élément**. Dans la boîte de dialogue **Ajouter un nouvel élément**, sélectionnez l'icône **Rapport**, puis cliquez sur **Ajouter**.

   Le Concepteur de rapports s'ouvre dans l'environnement de développement et un fichier de rapport (.rdlc) est ajouté au projet.
3. Faites glisser les éléments de rapport de la **Boîte à outils** vers la disposition de rapport et placez-les comme vous le souhaitez.
4. Faites glisser les champs de la fenêtre **Sources de données** vers les éléments de rapport dans la disposition de rapport.

Étapes suivantes de l'utilisation de ReportViewer et du Concepteur de rapports

Pour	Consultez
Ajouter et configurer un contrôle ReportViewer	Adding and Configuring the ReportViewer Controls
Créer et afficher l'aperçu d'un rapport local	Creating Client Report Definition (.rdlc) Files
Ajouter des sources de données pour des rapports ReportViewer	Creating Data Sources for a ReportViewer Report
Définir la disposition d'un rapport	Defining a Report Layout
Utiliser des expressions Visual Basic dans les rapports ReportViewer	Using Expressions in a ReportViewer Report
Placer du code personnalisé dans un rapport	Adding Custom Code to a ReportViewer Report
Distribuer des rapports et des contrôles ReportViewer dans le cadre de votre application	Deploying Reports and ReportViewer Controls
Imprimer un rapport	Printing Reports from ReportViewer

Pour plus d'instructions détaillées sur la façon de créer des rapports et de configurer des contrôles pour divers scénarios, consultez Samples and Walkthroughs.

Utilisation de Reporting Services dans les applications Visual Basic

Reporting Services est une technologie de création de rapports basée sur des serveurs ; elle est incluse avec SQL Server. Reporting Services inclut des fonctionnalités supplémentaires qui ne figurent pas dans les contrôles ReportViewer. Choisissez Reporting Services si vous avez besoin des fonctionnalités suivantes :

- Déploiement avec montée en puissance parallèle et traitement de rapports côté serveur qui améliorent les performances des rapports complexes, longs à s'exécuter et des systèmes présentant un gros volume d'activité.
- Traitement intégré de données et de rapports, avec la prise en charge de contrôles de rapports personnalisés et de formats de sortie élaborés.
- Traitement de rapports planifié afin que vous puissiez spécifier exactement quand les rapports sont exécutés.
- Distribution de rapports à des abonnés par messagerie électronique ou sur des emplacements de partage de fichiers.
- Création de rapports ad hoc afin que les utilisateurs professionnels puissent créer des rapports selon leurs besoins.
- Abonnements pilotés par des données qui routent la sortie des rapports personnalisés vers une liste dynamique de destinataires.
- Extensions personnalisées pour le traitement de données, la remise des rapports, l'authentification personnalisée et le rendu des rapports.

Le serveur de rapports est implémenté comme service Web. Le code de votre application doit inclure des appels au service Web pour accéder aux rapports et aux autres métadonnées. Le service Web fournit l'accès complet par programme à une instance de serveur de rapports.

Reporting Services étant une technologie de création de rapports basée sur le Web, la visionneuse par défaut affiche des rapports qui sont restitués au format HTML. Si vous ne souhaitez pas utiliser le format HTML comme format de présentation de rapport par défaut, vous devez écrire à une visionneuse de rapports personnalisée pour votre application.

Pour en savoir plus sur Reporting Services, consultez SQL Server Reporting Services dans la documentation en ligne de SQL Server 2005 et Didacticiels de Reporting Services.

Création de rapports dans Visual Studio pour Reporting Services

Pour générer des rapports qui s'exécutent sur un serveur de rapports, vous devez créer des fichiers de définition de rapport (.rdl) dans Visual Studio à l'aide de Business Intelligence Development Studio, qui est inclus avec SQL Server 2005.

Remarque
SQL Server 2005 doit être installé pour que vous puissiez utiliser SQL Server Reporting Services et Business Intelligence Development Studio.

Business Intelligence Development Studio ajoute des modèles de projet qui sont spécifiques aux composants SQL Server. Pour créer des rapports, vous avez le choix entre les modèles **Projet Report Server** ou **Assistant Projet Report Server**. Vous pouvez spécifier des requêtes et des connexions à la source de données pour divers types de source de données, tels que SQL Server, Oracle, Analysis Services, XML et SQL Server Integration Services. Les onglets **Données**, **Disposition** et **Aperçu** vous permettent de définir des données, de créer une disposition de rapport et d'afficher l'aperçu d'un rapport dans le même espace de travail.

Les définitions de rapports que vous générez pour le contrôle ou le serveur de rapports peuvent être réutilisées dans l'une ou l'autre de ces technologies. Pour plus d'informations, consultez Converting RDL and RDLC Files.

Pour créer un rapport qui s'exécute sur un serveur de rapports

1. Dans le menu **Fichier**, choisissez **Nouveau**.

   La boîte de dialogue **Nouveau projet** s'affiche.

2. Dans le volet **Types de projet**, cliquez sur **Projets Business Intelligence**.

3. Dans le volet Modèles, sélectionnez **Projet Report Server** ou **Assistant Projet Report Server**.

Utilisation conjointe de ReportViewer et de SQL Server Reporting Services

Les contrôles ReportViewer et SQL Server 2005 Reporting Services peuvent être utilisés conjointement dans la même application.

- Le contrôle ReportViewer fournit une visionneuse qui est utilisée pour afficher des rapports dans votre application.
- Reporting Services fournit les rapports et exécute tout le traitement sur un serveur distant.

Le contrôle ReportViewer peut être configuré pour afficher des rapports qui sont stockés et traités sur un serveur de rapports Reporting Services distant. Ce type de configuration est appelé *mode de traitement à distance*. En mode de traitement à distance, le contrôle demande un rapport qui est stocké sur un serveur de rapports distant. Le serveur de rapports exécute le traitement de tous les rapports, le traitement des données et le rendu des rapports. Un rapport fini et rendu est retourné au contrôle et affiché dans la zone d'affichage.

Les rapports qui s'exécutent sur un serveur de rapports prennent en charge des formats d'exportation supplémentaires, ont une implémentation du paramétrage des rapports différente, utilisent les types de source de données pris en charge par le serveur de rapports et sont accessibles via le modèle d'autorisation par rôle sur le serveur de rapports.

Pour utiliser le mode de traitement à distance, spécifiez l'URL et le chemin d'accès à un rapport serveur lors de la configuration du contrôle ReportViewer. Pour plus d'informations sur la configuration des contrôles et sur la comparaison entre les rapports serveurs et les rapports clients, consultez Configuring ReportViewer for Remote Processing.

Voir aussi

Référence

Microsoft.Reporting.WebForms
Microsoft.Reporting.WinForms

Autres ressources

ReportViewer Web Server and Windows Forms Controls
Creating Client Report Definition (.rdlc) Files
Introducing Reporting Services Programming
Présentation des fonctions de création, de gestion et de remise de rapports
Utilisation de données dans une mise en page de rapport
Utilisation des données dans Reporting Services

Personnalisation des projets et extension de My avec Visual Basic

Vous pouvez personnaliser des modèles de projet pour fournir des objets **My** supplémentaires. Ainsi, il est plus facile pour les autres développeurs de rechercher et d'utiliser vos objets.

En général, ces types de personnalisations sont particulièrement utiles pour les compléments.

Dans cette section

Extension du modèle d'application Visual Basic

Décrit comment spécifier vos propres extensions au modèle d'application en substituant les membres de la classe WindowsFormsApplicationBase.

Personnalisation de la disponibilité ou non des objets dans My

Décrit comment contrôler les objets **My** qui sont activés en définissant la constante de compilation conditionnelle _MYTYPE de votre projet.

Rubriques connexes

Développement avec My

Décrit quels objets **My** sont disponibles dans les différents types de projet par défaut.

Vue d'ensemble du modèle d'application Visual Basic

Décrit le modèle Visual Basic pour contrôler le comportement des applications Windows Forms.

Comment My dépend du type de projet

Décrit quels objets **My** sont disponibles dans les différents types de projet par défaut.

Vue d'ensemble de la compilation conditionnelle

Décrit comment le compilateur utilise la compilation conditionnelle pour sélectionner des rubriques spécifiques de code à compiler tout en excluant d'autres rubriques.

My.Application, objet

Décrit l'objet **My** qui fournit des propriétés, des méthodes et des événements en rapport avec l'application en cours.

Voir aussi

Autres ressources

Fonctionnalités plus poussées de Visual Basic

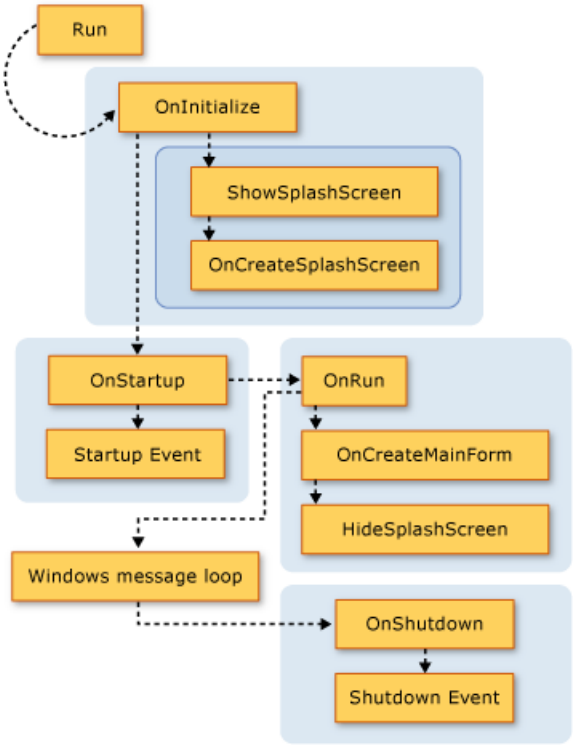
Extension du modèle d'application Visual Basic

Vous pouvez ajouter des fonctionnalités au modèle d'application en substituant les membres **Overridable** de la classe WindowsFormsApplicationBase. Cette technique permet de personnaliser le comportement du modèle d'application et d'ajouter des appels à vos propres méthodes au moment du démarrage ou de l'arrêt de l'application.

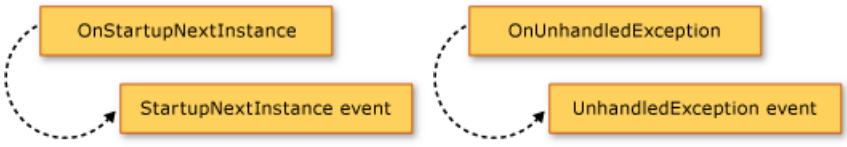
Vue d'ensemble visuelle du modèle d'application

Cette section présente visuellement la séquence des appels de fonction dans le modèle d'application Visual Basic. La section suivante décrit en détail l'objectif de chaque fonction.

Le graphique suivant illustre la séquence des appels du modèle d'application dans une application Windows Forms Visual Basic normale. La séquence commence lorsque la procédure `Sub Main` appelle la méthode Run.



Le modèle d'application Visual Basic fournit également les événements StartupNextInstance et UnhandledException. Les graphiques suivants illustrent le mécanisme de déclenchement de ces événements.



Substitution des méthodes de base

La My.Application.Run, méthode définit l'ordre dans lequel les méthodes **Application** s'exécutent. Par défaut, la procédure `Sub Main` d'une application Windows Forms appelle la méthode **Run**.

Si l'application est une application normale (application multi-instances), ou la première instance d'une application à instance unique, la méthode **Run** exécute les méthodes **Overridable** dans l'ordre suivant :

1. OnInitialize. Par défaut, cette méthode définit les styles visuels, les styles d'affichage de texte et l'entité de sécurité actuelle pour le thread de l'application principale (si l'application utilise l'authentification Windows) et appelle **ShowSplashScreen** si ni `/nosplash` ni `-nosplash` n'est utilisé comme un argument de ligne de commande.

   La séquence de démarrage de l'application est annulée si cette fonction retourne **False**. Cela peut être utile si l'application ne doit pas s'exécuter dans certaines circonstances.

   La méthode **OnInitialize** appelle les méthodes suivantes :

   a. ShowSplashScreen. Détermine si l'application a un écran de démarrage défini et, le cas échéant, l'affiche dans un

thread séparé.

La méthode **ShowSplashScreen** contient le code qui affiche l'écran de démarrage pendant au moins le nombre de millisecondes spécifié par la propriété MinimumSplashScreenDisplayTime. Pour utiliser ces fonctionnalités, vous devez ajouter l'écran de démarrage à votre application à l'aide du **Concepteur de projets** (qui affecte à la propriété **My.Application.MinimumSplashScreenDisplayTime** une durée de deux secondes) ou définir la propriété **My.Application.MinimumSplashScreenDisplayTime** dans une méthode qui substitue la méthode **OnInitialize** ou OnCreateSplashScreen. Pour plus d'informations, consultez My.Application.MinimumSplashScreenDisplayTime, propriété.

b. **OnCreateSplashScreen**. Permet à un concepteur d'émettre du code qui initialise l'écran de démarrage.

Par défaut, cette méthode n'effectue aucune opération. Si vous sélectionnez un écran de démarrage pour votre application dans le **Concepteur de projets** Visual Basic, le concepteur substitue la méthode **OnCreateSplashScreen** par une méthode qui affecte à la propriété SplashScreen une nouvelle instance du formulaire de l'écran de démarrage.

2. OnStartup. Fournit un point d'extensibilité pour déclencher l'événement **Startup**. La séquence de démarrage de l'application s'arrête si cette fonction retourne **False**.

   Par défaut, cette méthode déclenche l'événement Startup. Si le gestionnaire d'événements affecte à la propriété Cancel de l'argument d'événement la valeur `True`, la méthode retourne `False` pour annuler le démarrage de l'application.

3. OnRun. Fournit le point de démarrage à partir duquel l'application principale est prête à s'exécuter, une fois l'initialisation effectuée.

   Par défaut, avant qu'elle n'entre la boucle de message Windows Forms, cette méthode appelle les méthodes **OnCreateMainForm** (pour créer le formulaire principal de l'application) et **HideSplashScreen** (pour fermer l'écran de démarrage) :

   a. OnCreateMainForm. Permet à un concepteur d'émettre du code qui initialise le formulaire principal.

   Par défaut, cette méthode n'effectue aucune opération. Toutefois, lorsque vous sélectionnez un formulaire principal pour votre application dans le **Concepteur de projets** Visual Basic, le concepteur substitue la méthode **OnCreateMainForm** par une méthode qui affecte à la propriété MainForm une nouvelle instance du formulaire principal.

   b. HideSplashScreen. Si l'application a un écran de démarrage défini et ouvert, cette méthode ferme l'écran de démarrage.

   Par défaut, cette méthode ferme l'écran de démarrage.

4. OnStartupNextInstance. Permet de personnaliser le comportement d'une application à instance unique lorsqu'une autre instance de l'application démarre.

   Par défaut, cette méthode déclenche l'événement StartupNextInstance.

5. OnShutdown. Fournit un point d'extensibilité pour déclencher l'événement **Shutdown**. Cette méthode ne s'exécute pas si une exception non gérée se produit dans l'application principale.

   Par défaut, cette méthode déclenche l'événement Shutdown.

6. OnUnhandledException. Exécuté si une exception non gérée se produit dans l'une des méthodes répertoriées ci-dessus.

   Par défaut, cette méthode déclenche l'événement UnhandledException tant qu'un débogueur n'est pas rattaché et que l'application gère l'événement **UnhandledException**.

Si l'application est une application à instance unique, et qu'elle est déjà en cours d'exécution, son instance suivante appelle la méthode **OnStartupNextInstance** de l'instance d'origine de l'application, puis se ferme.

Le constructeur **WindowsFormsApplicationBase** appelle la propriété UseCompatibleTextRendering pour déterminer le moteur de rendu de texte à utiliser pour les formulaires de l'application. Par défaut, la propriété **UseCompatibleTextRendering** retourne **False**, ce qui indique que le moteur de rendu de texte GDI doit être utilisé (il s'agit de la valeur par défaut dans Visual Basic 2005). Vous pouvez substituer la propriété **UseCompatibleTextRendering** pour retourner **True**, indiquant ainsi que le moteur de rendu de texte GDI+ doit être utilisé (il s'agit de la valeur par défaut dans Visual Basic .NET 2002 et Visual Basic .NET 2003).

Configuration de l'application

Dans le modèle d'application Visual Basic, la classe **WindowsFormsApplicationBase** fournit des propriétés protégées qui configurent l'application. Ces propriétés doivent être définies dans le constructeur de la classe d'implémentation.

Dans un projet Windows Forms par défaut, le **Concepteur de projets** crée du code pour définir les propriétés avec les paramètres du concepteur. Les propriétés sont utilisées uniquement lorsque le démarrage de l'application est en cours ; il ne sert à rien de les utiliser après le démarrage de l'application.

Propriété	Détermine	Définition avec le Concepteur de projets
IsSingleInstance	Si l'application s'exécute comme une application à instance unique ou comme une application multi-instances.	Comment : spécifier le comportement d'instanciation pour une application
EnableVisualStyles	Si l'application utilise des styles visuels XP.	Comment : activer des styles visuels
SaveMySettingsOnExit	Si l'application enregistre automatiquement les modifications des paramètres utilisateur lorsqu'elle se ferme.	Activez la case à cocher **Enregistrer My.Settings lors de l'arrêt** dans le volet **Application** du **Concepteur de projets**.
ShutdownStyle	Ce qui provoque l'arrêt de l'application, par exemple lorsque le formulaire principal ou tous les formulaires se ferment.	Comment : spécifier le comportement d'arrêt pour une application

Voir aussi

Référence

My.Application, objet
My.Application.Startup, événement
My.Application.StartupNextInstance, événement
My.Application.UnhandledException, événement
My.Application.Shutdown, événement
My.Application.NetworkAvailabilityChanged, événement
WindowsFormsApplicationBase

Concepts

Vue d'ensemble du modèle d'application Visual Basic

Personnalisation de la disponibilité ou non des objets dans My

Cette rubrique décrit comment vous pouvez contrôler les objets **My** qui sont activés en définissant la constante de compilation conditionnelle **_MYTYPE** de votre projet. L'environnement de développement intégré (IDE, Integrated Development Environment) de Visual Studio conserve la constante de compilation conditionnelle **_MYTYPE** pour un projet synchronisé avec le type du projet.

Valeurs _MYTYPE prédéfinies

Vous devez utiliser l'option du compilateur **/define** pour définir la constante de compilation conditionnelle **_MYTYPE**. Lorsque vous spécifiez votre propre valeur pour la constante **_MYTYPE**, vous devez entourer la valeur de chaîne de barres obliques inverses/guillemets (\"). Par exemple, vous pouvez utiliser :

```
/define:_MYTYPE=\"WindowsForms\"
```

Ce tableau affiche la constante de compilation conditionnelle **_MYTYPE** affectée à plusieurs types de projet.

Type de projet	Valeur _MYTYPE
Bibliothèque de classes	"Windows"
Application console	"Console"
Web	"Web"
Bibliothèque de contrôles Web	"WebControl"
Application Windows	"WindowsForms"
Application Windows, lorsque vous démarrez avec un **Sub Main** personnalisé	"WindowsFormsWithCustomSubMain"
Bibliothèque de contrôles Windows	"Windows"
Service Windows	"Console"
Vide	"Empty"

Remarque

Toutes les comparaisons de chaînes de la compilation conditionnelle respectent les majuscules et les minuscules, quelle que soit la manière dont l'instruction **Option Compare** est définie.

Constantes de compilation _MY dépendantes

La constante de compilation conditionnelle **_MYTYPE** contrôle à son tour les valeurs de plusieurs autres constantes de compilation **_MY** :

_MYTYPE	_MYAPPLICATIONTYPE	_MYCOMPUTERTYPE	_MYFORMS	_MYUSERTYPE	_MYWEBSERVICES
"Console"	"Console"	"Windows"	Indéfini	"Windows"	TRUE
"Custom"	Indéfini	Indéfini	Indéfini	Indéfini	Indéfini
"Vide"	Indéfini	Indéfini	Indéfini	Indéfini	Indéfini
"Web"	Indéfini	"Web"	FALSE	"Web"	FALSE

"WebControl"	Indéfini	"Web"	FALSE	"Web"	TRUE
"Windows" ou ""	"Windows"	"Windows"	Indéfini	"Windows"	TRUE
"WindowsForms"	"WindowsForms"	"Windows"	TRUE	"Windows"	TRUE
"WindowsFormsWithCustomSubMain"	"Console"	"Windows"	TRUE	"Windows"	TRUE

Par défaut, les constantes de compilation conditionnelle non définies ont la valeur **FALSE**. Vous pouvez spécifier des valeurs pour les constantes non définies lors de la compilation de votre projet pour substituer le comportement par défaut.

Remarque
Lorsque **_MYTYPE** a la valeur "Custom", le projet contient l'espace de noms **My** mais ne contient pas d'objets. Toutefois, si vous affectez la valeur "Vide" à **_MYTYPE**, le compilateur ne pourra pas ajouter l'espace de noms **My** et ses objets.

Ce tableau décrit les effets des valeurs prédéfinies des constantes de compilation **_MY**.

Constante	Signification
_MYAPPLICATIONTYPE	Active **My.Application**, si la constante est "Console", Windows" ou "WindowsForms" : • La version "Console" dérive de ConsoleApplicationBase et a moins de membres que la version "Windows". • La version "Windows" dérive de ApplicationBase et a moins de membres que la version "WindowsForms". • La version "WindowsForms" de **My.Application** dérive de WindowsFormsApplicationBase. Si la constante **TARGET** est définie pour être "winexe", la classe inclut une méthode **Sub Main**.
_MYCOMPUTERTYPE	Active **My.Computer**, si la constante est "Web" ou "Windows" : • La version "Web" dérive de ServerComputer et a moins de membres que la version "Windows". • La version "Windows" de **My.Computer** dérive de Computer.
_MYFORMS	Active **My.Forms**, si la constante est **TRUE**.
_MYUSERTYPE	Active **My.User**, si la constante est "Web" ou "Windows" : • La version "Web" de **My.User** est associée à l'identité de l'utilisateur de la demande HTTP actuelle. • La version "Windows" de **My.User** est associée à l'entité de sécurité du thread actuel.
_MYWEBSERVICES	Active **My.WebServices**, si la constante est **TRUE**.
_MYTYPE	Active **My.Log**, **My.Request** et **My.Response**, si la constante est "Web".

Voir aussi

Référence

/define (Visual Basic)
My.Application, objet
My.Computer, objet
My.Forms, objet
My.Log, objet
My.Request, objet
My.Response, objet

My.User, objet
My.WebServices, objet

Concepts

Comment My dépend du type de projet
Vue d'ensemble de la compilation conditionnelle

Guide de programmation Visual Basic

À l'instar de n'importe quel langage de programmation récent, Visual Basic prend en charge plusieurs constructions de programmation et éléments de langage classiques. Ce guide passe en revue tous les éléments essentiels de la programmation avec Visual Basic.

Dans cette section

Structure de programme et conventions de codage

Contient la documentation sur la structure de base et les conventions de code de Visual Basic, par exemple les conventions d'affectation de noms, les commentaires dans le code et les limites de Visual Basic.

Fonctionnalités du langage Visual Basic

Présente et décrit les composants essentiels de Visual Basic.

Programmation orientée objet dans Visual Basic

Traite de la programmation orientée objet, notamment des classes et interfaces, de la création d'objets, des événements et délégués, de l'héritage ainsi que des groupes d'objets.

COM Interop

Explique les problèmes d'interopérabilité associés à la création et à l'utilisation d'objets COM (Component Object Model) avec Visual Basic.

Multithreading dans Visual Basic

Montre comment écrire du code qui exécute plusieurs tâches à la fois pour améliorer les performances et la réactivité de vos applications.

Rubriques connexes

Référence Visual Basic

Fournit des informations de référence sur les différents aspects de la programmation Visual Basic.

Compilateur Visual Basic

Fournit des informations sur l'utilisation du compilateur de ligne de commande Visual Basic, les options du compilateur et l'outil de mise à niveau de mot clé.

Structure de programme et conventions de codage

Cette section présente la structure du programme Visual Basic classique, offre un programme Visual Basic simple, « Hello World », et décrit les conventions de code Visual Basic. Les conventions de code sont des suggestions qui reposent non pas sur la logique d'un programme mais sur sa structure et son aspect physiques. Le respect de ces conventions facilite la lecture, la compréhension et la maintenance du code. Les conventions de code incluent notamment les éléments suivants :

- formats normalisés d'étiquetage et de commentaire du code ;
- indications relatives à l'espacement, au format et à la mise en retrait du code ;
- conventions d'affectation de noms à des objets, variables et procédures.

Les rubriques suivantes présentent un ensemble d'indications de programmation pour programmes Visual Basic, ainsi que des exemples de bonne utilisation.

Dans cette section

Structure d'un programme Visual Basic

Propose une vue d'ensemble des éléments qui composent un programme Visual Basic.

Version Visual Basic de Hello World!

Explique, étape par étape, comment recréer un programme « Hello World » classique reproduit en Visual Basic.

Procédure Main dans Visual Basic

Décrit la procédure qui sert de point de départ et de contrôle général de l'application.

Conventions d'affectation de noms Visual Basic

Inclut des indications générales pour les procédures d'affectation de noms, les constantes, variables, arguments et objets.

Conventions de codage Visual Basic

Étudie les indications utilisées dans le développement des exemples de cette documentation.

Compilation conditionnelle (Visual Basic)

Fournit des liens vers des rubriques relatives à la compilation sélective de blocs de code particuliers tout en indiquant au compilateur d'en ignorer d'autres.

Comment : diviser et combiner des instructions dans le code

Indique comment répartir les instructions longues sur plusieurs lignes et associer des instructions courtes sur une même ligne.

Comment : étiqueter des instructions

Illustre comment marquer une ligne de code afin de l'identifier pour être utilisée avec des instructions telles que **On Error Goto**.

Caractères spéciaux dans le code

Montre comment et où utiliser des caractères non numériques et non alphabétiques.

Commentaires dans le code

Indique comment ajouter des commentaires descriptifs à votre code.

Restrictions liées à Visual Basic

Aborde le problème de la suppression des limites connues du codage dans Visual Basic.

Rubriques connexes

Conventions typographiques

Propose des conventions de codage standard pour Visual Basic.

Modification de texte, de code et de balisage

Présente l'éditeur de code Visual Studio.

Structure d'un programme Visual Basic

Un programme Visual Basic est développé à partir de blocs de construction standard. Une *solution* comprend un ou plusieurs projets. Un *projet*, à son tour, peut contenir un ou plusieurs assemblys. Chaque *assembly* est compilé à partir d'un ou de plusieurs fichiers sources. Un *fichier source* fournit la définition et l'implémentation des classes, structures, modules et interfaces qui contiennent l'ensemble de votre code.

Éléments de programmation au niveau du fichier

Lorsque vous lancez un projet ou un fichier et ouvrez l'éditeur de code, vous constatez que figurent déjà dans cette fenêtre quelques lignes de code disposées dans le bon ordre. Tout code que vous créez doit respecter l'ordre suivant :

1. instructions **Option** ;
2. instructions **Imports** ;
3. instructions **Namespace** et éléments au niveau de l'espace de noms.

Si vous entrez les instructions dans un ordre différent, vous risquez de créer des erreurs de compilation.

Un programme peut également contenir des instructions de compilation conditionnelles. Vous pouvez les intercaler dans le fichier source entre les instructions de la séquence précédente.

Instructions Options

Les instructions **Option** définissent les règles à suivre pour l'écriture du code ultérieur, ce qui permet d'éviter les erreurs de syntaxe et de logique. L'instruction Option Explicit, instruction (Visual Basic) vérifie que toutes les variables sont déclarées et orthographiées correctement, ce qui réduit le temps de débogage. L'instruction Option Strict contribue à réduire les erreurs de logique et la perte de données susceptibles de se produire lorsque vous travaillez avec des variables de types de données différents. L'instruction Option Compare spécifie la manière dont les chaînes sont comparées les unes aux autres, en fonction de leurs valeurs **Binary** ou **Text**.

Instructions Imports

Vous pouvez inclure une instruction Imports pour importer des noms définis à l'extérieur de votre projet. Une instruction **Imports** permet à votre code de référencer des classes et d'autres types définis à l'intérieur de l'espace de noms importé sans devoir les qualifier. Vous pouvez utiliser autant d'instructions **Imports** que nécessaire. Pour plus d'informations, consultez Références et l'instruction Imports.

Instructions Namespace

Les espaces de noms vous aident à organiser et à classer vos éléments de programmation pour faciliter leur regroupement et leur accès. Vous utilisez l'instruction Namespace pour classer les instructions suivantes dans un espace de noms particulier. Pour plus d'informations, consultez Espaces de noms dans Visual Basic.

Instructions de compilation conditionnelles

Les instructions de compilation conditionnelles peuvent apparaître quasiment n'importe où dans votre fichier source. Elles provoquent l'inclusion ou l'exclusion de parties de votre code au moment de la compilation, en fonction de certaines conditions. Vous pouvez également les utiliser pour déboguer votre application, car le code conditionnel s'exécute uniquement en mode débogage. Pour plus d'informations, consultez Vue d'ensemble de la compilation conditionnelle.

Éléments de programmation au niveau de l'espace de noms

Les classes, les structures et les modules contiennent l'ensemble du code de votre fichier source. Ce sont des éléments *au niveau de l'espace de noms*, qui peuvent apparaître dans un espace de noms ou au niveau du fichier source. Ils contiennent les déclarations de tous les autres éléments de programmation. Les interfaces, qui définissent les signatures d'élément mais ne fournissent aucune implémentation, apparaissent également au niveau du module. Pour plus d'informations sur les éléments au niveau du module, consultez les rubriques suivantes :

- Class, instruction (Visual Basic)
- Structure, instruction
- Module, instruction
- Interface, instruction (Visual Basic)

Les éléments de données au niveau de l'espace de noms sont les énumérations et les délégués.

Éléments de programmation au niveau du module

Les procédures, les opérateurs, les propriétés et les événements sont les seuls éléments de programmation qui peuvent contenir du code exécutable (instructions qui exécutent des opérations au moment de l'exécution). Ce sont les éléments *au niveau du module* de votre programme. Pour plus d'informations sur les éléments au niveau de la procédure, consultez les rubriques suivantes :

- Function, instruction (Visual Basic)
- Sub, instruction (Visual Basic)
- Declare, instruction
- Operator, instruction
- Property, instruction
- Event, instruction

Les éléments de données au niveau du module sont les variables, les constantes, les énumérations et les délégués.

Éléments de programmation au niveau de la procédure

La majeure partie du contenu des éléments *au niveau de la procédure* sont des instructions exécutables qui constituent le code d'exécution de votre programme. Tout le code exécutable doit être contenu dans une procédure (**Function**, **Sub**, **Operator**, **Get**, **Set**, **AddHandler**, **RemoveHandler**, **RaiseEvent**). Pour plus d'informations, consultez Instructions exécutables.

Les éléments de données au niveau de la procédure se limitent aux variables et aux constantes locales.

Procédure Main

La procédure `Main` est le premier code à exécuter lorsque votre application a été chargée. `Main` sert de point de départ et de contrôle général pour votre application. Il existe quatre types de `Main` :

- `Sub Main()`
- `Sub Main(ByVal cmdArgs() As String)`
- `Function Main() As Integer`
- `Function Main(ByVal cmdArgs() As String) As Integer`

`Sub Main()` est le type le plus courant de cette procédure. Pour plus d'informations, consultez Procédure Main dans Visual Basic.

Voir aussi

Concepts

Version Visual Basic de Hello World!
Procédure Main dans Visual Basic
Conventions d'affectation de noms Visual Basic
Restrictions liées à Visual Basic

Version Visual Basic de Hello World!

Le programme de console suivant est la version Visual Basic du programme classique « Hello, World! », qui affiche la chaîne « Hello, World! ».

VB

```
' A "Hello, World!" program in Visual Basic.
Module Hello
  Sub Main()
      MsgBox("Hello, World!") ' Display message on computer screen.
  End Sub
End Module
```

Les points importants de ce programme sont les suivants :

- les commentaires ;
- la procédure `Main` ;
- les entrées et les sorties ;
- la compilation et l'exécution.

Commentaires

La première ligne de l'exemple contient un commentaire :

VB

```
' A "Hello, World!" program in Visual Basic.
```

Le guillemet simple (') signifie que le reste de la ligne constitue un commentaire et, en tant que tel, sera ignoré par le compilateur. Vous pouvez convertir une ligne complète en commentaire ou ajouter un commentaire à la fin d'une autre instruction, comme dans l'exemple qui suit :

VB

```
MsgBox("Hello, World!") ' Display message on computer screen.
```

Procédure Main

Toutes les applications Visual Basic doivent contenir une procédure appelée `Main`. Cette procédure sert de point de départ et de contrôle général de l'application. Elle est appelée lors du chargement du module.

Il existe quatre types de `Main` :

- `Sub Main()`
- `Sub Main(ByVal cmdArgs() As String)`
- `Function Main() As Integer`
- `Function Main(ByVal cmdArgs() As String) As Integer`

`Sub Main()` est le type le plus courant de cette procédure. À moins que vous ne créiez une application Windows Forms, vous devez écrire la procédure `Main` pour les applications qui s'exécutent seules. Pour plus d'informations, consultez Procédure Main dans Visual Basic.

Entrées et sorties

Cet exemple utilise la bibliothèque Visual Basic Runtime standard, disponible par l'intermédiaire de l'espace de noms Microsoft.VisualBasic. Si vous compilez le programme dans l'environnement de développement intégré (IDE, Integrated

Development Environment), vous pouvez utiliser toutes les procédures et propriétés de **Microsoft.VisualBasic** sans devoir l'importer. Si vous effectuez une compilation à partir de la ligne de commande, vous devez utiliser l'Imports, instruction dans votre code source, ou l'option du compilateur de ligne de commande /imports (Visual Basic), pour que les membres **Microsoft.VisualBasic** soient disponibles pour votre programme.

La procédure `Main` appelle la fonction MsgBox, fonction (Visual Basic) pour afficher un message contenant la chaîne « Hello, World! » :

VB

```
MsgBox("Hello, World!") ' Display message on computer screen.
```

Compilation et exécution

Vous pouvez compiler le programme « Hello, World! » en utilisant l'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio ou la ligne de commande.

Pour compiler et exécuter le programme à partir de la ligne de commande

1. Créez le fichier source à l'aide d'un éditeur de texte quelconque et enregistrez-le sous un nom tel que `Hello.vb`.
2. Pour appeler le compilateur, entrez la commande suivante :

   ```
   vbc Hello.vb
   ```

   Si votre fichier source ne contient pas d'instruction **Imports** pour l'espace de noms **Microsoft.VisualBasic**, vous pouvez inclure l'option du compilateur de ligne de commande **/imports** dans la commande `vbc` :

   ```
   vbc Hello.vb /imports:Microsoft.VisualBasic
   ```
3. Si votre programme est exempt d'erreurs de compilation, le compilateur crée un fichier `Hello.exe`.
4. Pour exécuter le programme, entrez la commande suivante :

   ```
   Hello
   ```

Vous pouvez également inclure l'option de compilateur de ligne de commande **/main** dans la commande `vbc` pour indiquer l'espace de noms et le module qui fournissent la déclaration `Main`.

Pour compiler et exécuter le programme à partir de l'environnement de développement intégré

1. Créez un projet d'application console Visual Basic.
2. Copiez le code dans le projet.
3. Sélectionnez la commande **Générer** appropriée dans le menu **Générer**, ou appuyez sur F5 pour la génération et l'exécution (équivaut à l'option **Démarrer** dans le menu **Déboguer**).

Pour plus d'informations sur le compilateur Visual Basic et ses options, consultez
Génération à partir de la ligne de commande (Visual Basic).

Voir aussi

Référence

Structure d'un programme Visual Basic
Imports, instruction
/imports (Visual Basic)
MsgBox, fonction (Visual Basic)
/main
Microsoft.VisualBasic

Concepts

Procédure Main dans Visual Basic

Autres ressources

Génération à partir de la ligne de commande (Visual Basic)

Procédure Main dans Visual Basic

Toutes les applications Visual Basic doivent contenir une procédure appelée `Main`. Cette procédure sert de point de départ et de contrôle général de l'application. .NET Framework appelle votre procédure `Main` lorsqu'il a chargé votre application et qu'il est prêt à lui passer le contrôle. À moins que vous ne créiez une application Windows Forms, vous devez écrire la procédure `Main` pour les applications qui s'exécutent seules.

`Main` contient le code qui s'exécute en premier. Dans `Main`, vous pouvez identifier le premier formulaire à être chargé lors du démarrage du programme, savoir si une copie de votre application s'exécute déjà sur le système, définir un ensemble de variables pour votre application ou ouvrir une base de données requise par l'application.

Conditions de la procédure Main

Un fichier qui s'exécute seul (généralement avec l'extension .exe) doit contenir une procédure `Main`. Une bibliothèque (par exemple avec l'extension .dll) ne s'exécute pas toute seule et ne requiert pas de procédure `Main`. Les conditions relatives à ces différents types de projets que vous pouvez créer sont les suivantes :

- Les applications console s'exécutent toutes seules et vous devez fournir au moins une procédure `Main`. .
- Les applications Windows Forms s'exécutent toutes seules. Toutefois, le compilateur Visual Basic génère automatiquement une procédure `Main` dans une telle application et vous n'avez pas besoin d'en écrire une.
- Les bibliothèques ne requièrent pas de procédure `Main`. Celles-ci comprennent les bibliothèques de contrôles Windows et les bibliothèques de contrôles Web. Les applications Web sont déployées comme des bibliothèques de classes.

Déclaration de la procédure Main

Il existe quatre méthodes possibles pour déclarer la procédure `Main` : Elle peut prendre ou non des arguments, retourner ou non une valeur.

Remarque
Si vous déclarez la procédure `Main` dans une classe, vous devez utiliser le mot clé **Shared**. Dans un module, la procédure `Main` n'a pas besoin d'être partagée (**Shared**).

- La méthode la plus simple consiste à déclarer une procédure **Sub** que ne prend pas d'arguments et ne retourne pas de valeur.

```
Module mainModule
    Sub Main()
        MsgBox("The Main procedure is starting the application.")
        ' Insert call to appropriate starting place in your code.
        MsgBox("The application is terminating.")
    End Sub
End Module
```

- La procédure `Main` peut également retourner une valeur **Integer**, que le système d'exploitation utilise comme code de sortie du programme. D'autres programmes peuvent tester ce code en examinant la valeur ERRORLEVEL Windows. Pour retourner un code de sortie, vous devez déclarer `Main` comme une procédure **Function** et non comme une procédure **Sub**.

```
Module mainModule
    Function Main() As Integer
        MsgBox("The Main procedure is starting the application.")
        Dim returnValue As Integer = 0
        ' Insert call to appropriate starting place in your code.
        ' On return, assign appropriate value to returnValue.
        ' 0 usually means successful completion.
        MsgBox("The application is terminating with error level " _
```

```
            & CStr(returnValue) & ".")
        Return returnValue
    End Function
End Module
```

- `Main` peut également prendre comme argument un tableau **String**. Chaque chaîne du tableau contient un des arguments de ligne de commande utilisés pour appeler le programme. Vous pouvez réaliser diverses actions en fonction de leurs valeurs.

```
Module mainModule
    Function Main(ByVal cmdArgs() As String) As Integer
        MsgBox("The Main procedure is starting the application.")
        Dim returnValue As Integer = 0
        ' See if there are any arguments.
        If cmdArgs.Length > 0 Then
            For argNum As Integer = 0 To UBound(cmdArgs, 1)
                ' Insert code to examine cmdArgs(argNum) and take
                ' appropriate action based on its value.
            Next argNum
        End If
        ' Insert call to appropriate starting place in your code.
        ' On return, assign appropriate value to returnValue.
        ' 0 usually means successful completion.
        MsgBox("The application is terminating with error level " _
            & CStr(returnValue) & ".")
        Return returnValue
    End Function
End Module
```

- Vous pouvez déclarer `Main` de sorte à examiner les arguments de ligne de commande, sans qu'elle ne retourne toutefois un code de sortie.

```
Module mainModule
    Sub Main(ByVal cmdArgs() As String)
        MsgBox("The Main procedure is starting the application.")
        Dim returnValue As Integer = 0
        ' See if there are any arguments.
        If cmdArgs.Length > 0 Then
            For argNum As Integer = 0 To UBound(cmdArgs, 1)
                ' Insert code to examine cmdArgs(argNum) and take
                ' appropriate action based on its value.
            Next argNum
        End If
        ' Insert call to appropriate starting place in your code.
        MsgBox("The application is terminating."
    End Sub
End Module
```

Voir aussi

Référence

Structure d'un programme Visual Basic
/main
Shared (Visual Basic)
MsgBox, fonction (Visual Basic)
Sub, instruction (Visual Basic)
Function, instruction (Visual Basic)
Integer, type de données (Visual Basic)
String, type de données (Visual Basic)

UBound, fonction (Visual Basic)
Length
Concepts
Version Visual Basic de Hello World!

Conventions d'affectation de noms Visual Basic

Lorsque vous nommez un élément de votre application Visual Basic, le premier caractère de ce nom doit être un caractère alphanumérique ou un trait de soulignement. Toutefois, notez que les noms commençant par un trait de soulignement ne sont pas conformes avec la Spécification CLS (Common Language Specification) (CLS).

Les suggestions suivantes s'appliquent à l'affectation de noms :

- Faites commencer chaque mot distinct d'un nom par une majuscule, par exemple `FindLastRecord` et `RedrawMyForm`.
- Faites commencer les noms de fonctions et de méthodes par un verbe, comme dans `InitNameArray` ou `CloseDialog`.
- Faites commencer les noms de classes, de structures, de modules et de propriétés par un nom, comme dans `EmployeeName` ou `CarAccessory`.
- Faites commencer les noms de l'interface par le préfixe « I », suivi d'un nom ou d'une expression nominale (par exemple, `IComponent`) ou d'un adjectif décrivant le comportement de l'interface (par exemple, `IPersistable`). N'utilisez pas le trait de soulignement et évitez autant que possible les abréviations, qui sont source de confusion.
- Faites commencer les noms de gestionnaires d'événements par un nom décrivant le type d'événements, suivi du suffixe « `EventHandler` », par exemple « `MouseEventHandler` ».
- Dans les noms de classes d'arguments d'événements, ajoutez le suffixe « `EventArgs` ».
- Si un événement contient une notion d'antériorité ou de postériorité, utilisez un préfixe décliné au présent ou au passé, par exemple « `ControlAdd` » ou « `ControlAdded` ».
- Pour les termes longs ou fréquemment utilisés, utilisez des abréviations afin d'en limiter la longueur, par exemple « HTML » au lieu de « Hypertext Markup Language ». D'une manière générale, les noms de variables d'une longueur supérieure à 32 caractères sont difficiles à lire sur un écran avec une résolution basse. Assurez-vous également que vos abréviations sont cohérentes dans toute l'application. En effet, le fait d'utiliser indifféremment « HTML » et « Hypertext Markup Language » ne peut être que source de confusion.
- Évitez d'utiliser dans une portée interne les mêmes noms que dans une portée externe. Des erreurs peuvent être générées en cas d'accès à la mauvaise variable. En cas de conflit entre une variable et le mot clé du même nom, vous devez identifier le mot clé en le faisant précéder de la bibliothèque de types appropriée. Par exemple, si l'une de vos variables porte le nom `Date`, vous ne pouvez utiliser la fonction **Date** intrinsèque qu'en appelant System.DateTime.Date.

Voir aussi

Concepts

Utilisation des mots clés comme noms d'éléments dans le code
Me, My, MyBase et MyClass dans Visual Basic
Noms d'éléments déclarés

Autres ressources

Structure de programme et conventions de codage
Référence Visual Basic

Utilisation des mots clés comme noms d'éléments dans le code

Tout élément de programme — tel qu'une variable, une classe ou un membre — peut avoir le même nom qu'un mot clé restreint. Vous pouvez, par exemple, créer une variable appelée `Loop`. Cependant, pour faire référence à votre version de variable — qui porte le même nom que le mot clé **Loop** restreint — vous devez soit la faire précéder d'une chaîne de qualification complète, soit la placer entre crochets (**[ ]**), comme le montrent les exemples suivants :

VB

```
' The following statement precedes Loop with a full qualification string.
sampleForm.Loop.Visible = True
' The following statement encloses Loop in square brackets.
 [Loop].Visible = True
```

Si vous ne procédez pas de l'une de ces façons, Visual Basic part du principe que le mot clé **Loop** intrinsèque est utilisé et il génère une erreur, comme ci-dessous :

```
' The following statement causes a compiler error.

Loop.Visible = True
```

Vous pouvez utiliser des crochets lorsque vous faites référence à des formulaires et des contrôles, et lorsque vous déclarez une variable ou définissez une procédure ayant le même nom qu'un mot clé restreint. Il peut être assez facile d'oublier de qualifier des noms ou d'inclure des crochets, et ainsi d'introduire des erreurs dans votre code et de le rendre difficile à lire. C'est la raison pour laquelle nous vous recommandons de ne pas utiliser de mots clés restreints comme noms d'éléments de programme. Cependant, si une version ultérieure de Visual Basic définit un nouveau mot clé qui entre en conflit avec un nom de formulaire ou de contrôle existant, vous pouvez utiliser cette technique lorsque vous effectuez une mise à jour de votre code afin de pouvoir utiliser la nouvelle version.

Remarque

Votre programme peut également contenir des noms d'éléments fournis par d'autres assemblys référencés. Si ces noms entr ent en conflit avec des mots clés restreints, le fait de les placer entre crochets force Visual Basic à les interpréter comme vos é léments définis.

Voir aussi

Concepts

Conventions d'affectation de noms Visual Basic

Autres ressources

Structure de programme et conventions de codage

Mots clés (Visual Basic)

Me, My, MyBase et MyClass dans Visual Basic

Les ressemblances apparentes entre **Me**, **My**, **MyBase** et **MyClass** en Visual Basic peuvent créer la confusion si vous abordez les concepts pour la première fois. Cette page décrit chacune de ces entités pour établir une distinction entre elles.

Me

Le mot clé **Me** permet de faire référence à l'instance spécifique d'une classe ou d'une structure dans laquelle le code s'exécute actuellement. **Me** se comporte comme une variable objet ou une variable structure faisant référence à l'instance actuelle. L'utilisation de **Me** est particulièrement utile pour le passage des informations concernant l'instance d'une classe ou d'une structure en cours d'exécution, à une procédure se trouvant dans une autre classe, une autre structure ou un autre module.

My

La fonction **My** fournit un accès facile et intuitif à plusieurs classes du .NET Framework, ce qui permet à l'utilisateur Visual Basic d'interagir avec l'ordinateur, l'application, les paramètres, les ressources, etc.

MyBase

Le mot clé **MyBase** se comporte comme une variable objet faisant référence à la classe de base de l'instance actuelle d'une classe. **MyBase** est couramment utilisé pour accéder aux membres de la classe de base qui sont substitués ou occultés dans une classe dérivée. **MyBase.New** permet d'appeler explicitement un constructeur de classe de base d'un constructeur de classe dérivé.

MyClass

Le mot clé **MyClass** se comporte comme une variable objet qui fait référence à l'instance actuelle d'une classe comme implémentée à l'origine. **MyClass** est semblable à **Me**, mais tous les appels de méthode effectués sur lui sont traités comme si la méthode était **NotOverridable**.

Voir aussi

Référence

MyBase
MyClass
Me

Concepts

Développement avec My
Éléments fondamentaux de l'héritage

Conventions de codage Visual Basic

Ces conventions sont utilisées par Microsoft pour développer des exemples et de la documentation. La spécification du langage Visual Basic ne définit aucune norme de codage.

- Les conventions de codage donnent au code un aspect homogène, afin que les lecteurs puissent se concentrer sur le contenu et non pas sur la disposition.
- Les conventions permettent aux lecteurs de comprendre le code plus rapidement, parce qu'ils peuvent faire des hypothèses selon leur expérience précédente.
- Les conventions facilitent la copie, la modification et la gestion du code.
- Les conventions décrivent les méthodes conseillées pour Visual Basic.

Discussion

Conventions d'affectation de noms

- Les indications concernant l'attribution d'un nom sont décrites dans Instructions de conception pour le développement de bibliothèques de classes.
- Vous ne devez pas modifier le nom des objets créés avec les outils du Concepteur de Visual Studio pour les rendre conformes aux indications.
- Utilisez les qualifications d'espace de noms au lieu d'ajouter des instructions Imports. Si un espace de noms est importé par défaut dans un projet, vous ne devez pas qualifier pleinement le code parce qu'il s'exécutera de façon non qualifiée avec Intellisense lors du copier-coller. Lorsque vous coupez de longues lignes de code pour en simplifier la lecture, les noms qualifiés risquent d'être coupés après le "." Par exemple :

```
Dim collection As System.Diagnostics. _
       InstanceDataCollectionCollection
```

- N'utilisez pas "My" ou "my" dans un nom de variable, car cela peut créer une confusion avec les objets **My**.

Conventions de disposition

Une bonne disposition utilise la mise en forme pour souligner la structure du code et simplifier la lecture du code.

- Utilisez la fonctionnalité Mise en forme automatique avec tabulation du code pour mettre le code en forme avec les paramètres par défaut (mise en retrait intelligente, mise en retrait de 4 caractères, enregistrement des onglets en tant qu'espaces). Pour plus d'informations, consultez Spécifique à VB, Basic, Éditeur de texte, boîte de dialogue Options.
- Mettez une seule instruction par ligne. N'utilisez pas les deux-points (:).
- Mettez une seule déclaration par ligne.
- Mettez en retrait les lignes de continuation en utilisant un taquet de tabulation.
- Laissez au moins une ligne vide entre les définitions de méthode et de propriété.

Conventions de commentaires

- Ne mettez pas de commentaires à la fin d'une ligne de code, mais plutôt sur une ligne séparée.
- Commencez le texte du commentaire par une lettre majuscule.
- Ajoutez un point à la fin du commentaire.
- Insérez un espace entre le délimiteur de commentaire (') et le texte de commentaire.

```
' Here is a comment.
```

- Ne créez pas de blocs d'astérisques mis en forme autour des commentaires.

Structure du programme

- Avec la méthode **Main**, utilisez la construction par défaut pour les nouvelles applications console et utilisez **My** pour les arguments de ligne de commande.

```
Sub Main()
    For Each argument As String In My.Application.CommandLineArgs
        ' Add code here to use the string variable.
    Next
End Sub
```

Indications concernant le langage
String, type de données

- Utilisez le signe & pour concaténer des chaînes :

```
MsgBox("hello" & vbCrLf & "goodbye")
```

- Pour ajouter des chaînes dans des boucles, utilisez l'objet StringBuilder :

```
Dim longString As New System.Text.StringBuilder
For count As Integer = 1 To 1000
    longString.Append(count)
Next
```

Type de données non signé

- Utilisez **Integer** plutôt que des types non signés sauf si vous donnez plus d'importance à la mémoire.

Tableaux

- Utilisez une syntaxe courte lorsque vous initialisez des tableaux sur la ligne de déclaration :

```
Dim letters() As String = {"a", "b", "c"}
```

Au lieu de :

```
Dim letters() As String = New String() {"a", "b", "c"}
```

- Mettez l'indicateur de tableau sur la variable, pas sur le type :

```
Dim letters() As String = {"a", "b", "c"}
```

Au lieu de :

```
Dim letters As String() = {"a", "b", "c"}
```

- Utilisez la syntaxe {} lorsque vous déclarez et initialisez des tableaux comportant des types de données de base :

```
Dim letters() As String = {"a", "b", "c"}
```

Au lieu de :

```
Dim letters(2) As String
letters(0) = "a"
letters(1) = "b"
letters(2) = "c"
```

Utilisation du mot clé "With"

Lorsque vous êtes confronté à une série d'appels à un objet, utilisez le mot clé **With**.

Déclaration de variables de boucle dans l'instruction For ou For Each

- Exemple :

```
For count As Integer = 0 To 2
    MsgBox(names(count))
Next
```

- Exemple :

```
For Each name As String In names
    MsgBox(name)
Next
```

Utilisation des instructions Try...Catch

- Utilisez toujours les instructions **Try...Catch** avec un objet qui implémente une méthode **Dispose**.
- N'utilisez pas l'instruction On Error Goto.

Utilisation de l'instruction Using

Si vous utilisez une instruction **Try...Catch** et si le seul code dans le bloc **Finally** est un appel à **Dispose**, utilisez l'instruction **Using** à la place.

Mot clé IsNot

Utilisez le mot clé **IsNot** plutôt que **Not ... Is Nothing**.

Instances de formulaires par défaut

Utilisez `Form1.ShowDialog` plutôt que `My.Forms.Form1.ShowDialog`.

Mot clé New

- Utilisez l'instanciation courte :

```
Dim employees As New Collection()
```

  Au lieu de :

```
Dim employees As Collection = New Collection()
```

- Utilisez le constructeur sans paramètre, puis définissez les propriétés à l'aide du mot clé **With** :

```
Dim orderLog As New EventLog()
With orderLog
    .Log = "Application"
    .Source = "Application Name"
    .MachineName = "Machine Name"
End With
```

Gestion des événements

- Utilisez **Handles** plutôt que **AddHandler** :

```
Private Sub MenuItem1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem1.Click
```

- Utilisez **AddressOf** et n'instanciez pas le délégué explicitement :

```
Dim closeItem As New MenuItem("Close", AddressOf MenuItem1_Click)
Me.MainMenu1.MenuItems.Add(closeItem)
```

- Lorsque vous définissez un événement, utilisez la syntaxe courte et laissez le compilateur définir le délégué :

```
Public Event WhatHappened(ByVal source As Object, _
    ByVal e As WhatHappenedEventArgs)
```

Utilisation des membres partagés

Membres **Shared** appelés à l'aide du nom de classe, ne provenant pas d'une variable d'instance.

MsgBox, fonction

Utilisez **MsgBox** au lieu de `MessageBox.Show` ou `Console.WriteLine`.

Utilisation de l'espace de noms My

Utilisez les fonctionnalités **My** plutôt que la bibliothèque de classes .NET Framework ou la bibliothèque runtime Visual Basic.

Utilisation des membres de la bibliothèque runtime Visual Basic

Utilisez la bibliothèque runtime Visual Basic plutôt que la bibliothèque de classes .NET Framework.

Indications concernant les exemples
Générales

- Suivez les indications de design dans Indications concernant le design pour les développeurs de bibliothèques de classes (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp).
- Spécifiez l'invite et le titre des appels **MsgBox**.
- Utilisez des fichiers de ressources quand cela est approprié.
- Utilisez **Option Strict On** soit dans chaque fichier, soit en tant que paramètre du projet.
- Procédez à la compilation après avoir activé tous les avertissements.
- Définissez un seul **Class**, **Structure** ou **Interface** dans un fichier.
- Utilisez le codage par défaut pour enregistrer les fichiers.

Localisation

- Utilisez la propriété `AutoSize` chaque fois que possible.
- Ne masquez pas et ne superposez pas les contrôles.
- N'alignez pas les contrôles pour créer une phrase.
- Ne construisez pas de chaînes en supprimant des caractères d'une autre chaîne.
- Utilisez des graphiques de culture neutre.
- Utilisez uniquement des polices Tahoma ou MS Sans Serif.

Accessibilité

- Utilisez les couleurs figurant dans l'onglet **Système** de la boîte de dialogue du sélecteur de couleurs.
- Utilisez des accélérateurs pour tous les menus, étiquettes, boutons, etc.
- Définissez les propriétés des contrôles comme indiqué dans le tableau suivant.

Propriété	**Paramètre**
AccessibleDescription	Description du contrôle.

AccessibleName	Nom du contrôle.
AccessibleRole	Par défaut ; ou bien vous pouvez redéfinir cette propriété si le contrôle a un autre rôle.
TabIndex	Classé dans un ordre logique.
Text	Tous les contrôles interactifs doivent avoir une touche d'accès rapide sur le clavier (raccourci).
Font size	Par défaut ; ou bien vous pouvez définir la taille de la police à 10 points ou plus.
Forecolor	Par défaut
BackColor	Par défaut
BackgroundImage	Par défaut

Security

Suivez les indications figurant dans Indications de codage sécurisé.

Voir aussi

Autres ressources

Instructions de conception pour le développement de bibliothèques de classes
Indications de codage sécurisé

Compilation conditionnelle (Visual Basic)

La *compilation conditionnelle* permet de compiler de façon sélective des blocs de code spécifiques tout en excluant d'autres blocs.

Dans cette section

Vue d'ensemble de la compilation conditionnelle

Explique à quoi sert la compilation conditionnelle et décrit les directives **#Const** et **#If...Then...#Else**.

Comment : déclarer des constantes de compilation conditionnelle

Explique comment déclarer des constantes de compilation conditionnelle dans la boîte de dialogue **Pages de propriétés**, sur la ligne de commande ou dans le code.

Comment : réduire et masquer des sections de code

Explique comment spécifier un bloc de code que vous pouvez développer ou réduire à l'aide de la fonctionnalité mode Plan de Visual Studio.

Rubriques connexes

Structure de programme et conventions de codage

Fournit quelques suggestions pour simplifier la lecture et la gestion de votre code.

Compilation en MSIL

Explique comment votre code source est traduit en langage MSIL (Microsoft Intermediate language) par le compilateur.

Compilation du MSIL en code natif

Fournit une vue d'ensemble du compilateur .NET Framework JIT (just-in-time) qui convertit le langage MSIL en code natif (spécifique au processeur).

Vue d'ensemble de la compilation conditionnelle

Il est possible d'utiliser la compilation conditionnelle pour sélectionner des rubriques spécifiques de code à compiler tout en excluant d'autres rubriques. Vous pouvez, par exemple, souhaiter écrire des instructions de débogage qui comparent la rapidité de plusieurs approches d'une même tâche de programmation ou encore localiser une application dans plusieurs langues. Les instructions de compilation conditionnelle sont conçues pour s'exécuter au moment de la compilation et non de l'exécution.

Déclarez une constante de compilation conditionnelle avec la directive **#Const** et indiquez les blocs de code qui doivent être compilés conditionnellement avec la directive **#If...Then...#Else**. Pour créer, par exemple, des versions en langue française et allemande de la même application à partir du même code source, incorporez des segments de code propres à la plate-forme dans des instructions **#If...Then** à l'aide des constantes prédéfinies `FrenchVersion` et `GermanVersion`. L'exemple suivant vous indique comment procéder :

VB

```
#If FrenchVersion Then
   ' <code specific to the French language version>.
#ElseIf GermanVersion Then
   ' <code specific to the German language version>.
#Else
        ' <code specific to other versions>.
#End If
```

Si vous attribuez la valeur **True** à la constante `FrenchVersion` au moment de la compilation, le code conditionnel de la version française est compilé. Si vous attribuez la valeur **True** à la constante `GermanVersion`, le compilateur utilise la version allemande. Si aucune des deux n'a la valeur **True**, c'est le code écrit dans le dernier bloc **Else** qui est exécuté.

Remarque

La saisie semi-automatique ne fonctionne pas lorsque vous modifiez du code et que vous utilisez des directives de compilation conditionnelles si le code ne fait pas partie du branchement actuel.

Voir aussi

Tâches

Comment : déclarer des constantes de compilation conditionnelle

Comment : réduire et masquer des sections de code

Référence

#Const, directive

#If...Then...#Else, directives

Autres ressources

Génération à partir de la ligne de commande (Visual Basic)

Comment : déclarer des constantes de compilation conditionnelle

Vous avez le choix entre trois méthodes pour définir des constantes de compilation conditionnelle :

- Dans le **Concepteur de projets**
- dans la ligne de commande lorsque vous utilisez le compilateur de ligne de commande ;
- dans votre code.

Les constantes de compilation conditionnelle sont dotées d'une portée spéciale et ne sont pas accessibles à partir du code standard. La portée d'une constante de compilation conditionnelle varie en fonction de la façon dont elle a été définie. Le tableau suivant répertorie la portée des constantes déclarées à l'aide de chacune des trois méthodes décrites ci-dessus.

Méthode de définition de la constante	**Portée de la constante**
Concepteur de projets	Publique pour tous les fichiers du projet.
Ligne de commande	Publique pour tous les fichiers passés au compilateur de ligne de commande.
Instruction **#Const** dans le code	Privée pour le fichier dans lequel elle est déclarée.

Pour définir des constantes dans le Concepteur de projets

- Avant de créer votre fichier exécutable, définissez des constantes dans le **Concepteur de projets** en suivant les étapes décrites dans Comment : modifier des propriétés de projet et des paramètres de configuration.

Pour définir des constantes dans la ligne de commande

- Utilisez le commutateur **/d** pour entrer des constantes de compilation conditionnelle, comme dans l'exemple suivant :

  ```
  vbc MyProj.vb /d:conFrenchVersion=-1:conANSI=0
  ```

  Aucun espace n'est nécessaire entre le commutateur **/d** et la première constante. Pour plus d'informations, consultez /define (Visual Basic).

  Les déclarations de ligne de commande se substituent aux déclarations entrées dans le **Concepteur de projets**, mais elles ne les suppriment pas. Les arguments définis dans le **Concepteur de projets** restent en application pour des compilations ultérieures.

  Lorsque vous écrivez des constantes dans le code lui-même, il n'existe aucune règle fixe quant à leur placement dans la mesure où leur portée correspond à tout le module dans lequel elles sont déclarées.

Pour déterminer des constantes dans votre code

- Placez les constantes dans le bloc de déclaration du module dans lequel elles sont utilisées. Cela contribue à conserver un code organisé et plus facile à lire.

Voir aussi

Tâches

Comment : réduire et masquer des sections de code

Référence

Constantes de compilation conditionnelle
/define (Visual Basic)
#If...Then...#Else, directives
#Const, directive

Concepts

Vue d'ensemble de la compilation conditionnelle

Autres ressources

Génération à partir de la ligne de commande (Visual Basic)

Comment : réduire et masquer des sections de code

La directive **#Region** vous permet de réduire et de masquer des sections de votre code dans des fichiers Visual Basic. La directive **#Region** vous permet de spécifier un bloc de code que vous pouvez développer ou réduire lorsque vous utilisez l'éditeur de code de Visual Studio. Cette fonctionnalité de masquage sélectif du code facilite la gestion et la lecture de vos fichiers. Pour plus d'informations, consultez Comment : structurer et masquer le code.

Les directives **#Region** prennent en charge la sémantique de bloc de code telle que **#If...#End If**. Cela signifie qu'elles ne peuvent pas commencer dans un bloc de code et se terminer dans un autre : le début et la fin doivent figurer dans le même bloc. Les directives **#Region** ne sont pas prises en charge dans les fonctions.

Pour réduire et masquer une section de code

- Placez la section de code entre les instructions **#Region** et **#End Region**, comme dans l'exemple suivant :

  VB

```
#Region "This is the code to be collapsed"
    Private components As System.ComponentModel.Container
    Dim WithEvents Form1 As System.Windows.Forms.Form

    Private Sub InitializeComponent()
        components = New System.ComponentModel.Container
        Me.Text = "Form1"
    End Sub
#End Region
```

  Il est possible d'utiliser plusieurs fois le bloc **#Region** dans un fichier de code. Ainsi, les utilisateurs peuvent définir leurs propres blocs de procédures et de classes qui peuvent, à leur tour, être réduits. Les blocs **#Region** peuvent également être imbriqués dans d'autres blocs **#Region**.

Remarque
Le masquage du code n'empêche pas sa compilation et n'affecte pas les instructions **#If...#End If**.

Voir aussi

Tâches

Comment : déclarer des constantes de compilation conditionnelle

Comment : structurer et masquer le code

Référence

#Region, directive

#If...Then...#Else, directives

Concepts

Vue d'ensemble de la compilation conditionnelle

Comment : diviser et combiner des instructions dans le code

Lorsque vous écrivez votre code, vous pouvez être amené à créer de longues instructions qui vous imposent un défilement horizontal dans l'éditeur de code. Même si cela n'affecte en rien l'exécution de votre code, cela en complique la lecture à l'écran, pour vous comme pour toute autre personne. Il est donc fortement recommandé de répartir cette instruction longue sur plusieurs lignes.

À l'inverse, vous pouvez être amené à consolider des instructions sur une seule ligne, notamment lorsque vous avez plusieurs instructions très courtes et que vous souhaitez garder de la place pour la suite. Cette fonctionnalité peut également être utile lorsqu'il s'agit d'organiser des variables ou des commandes dans un module.

Pour couper une même instruction en plusieurs lignes

- À l'endroit auquel vous voulez couper la ligne, utilisez le caractère de continuation de ligne, qui est un trait de soulignement (_). Le trait de soulignement doit être immédiatement précédé d'un espace et immédiatement suivi d'une marque de fin de ligne (retour chariot).

  Dans l'exemple ci-dessous, l'instruction est répartie sur quatre lignes à l'aide de caractères de continuation terminant toutes les lignes à l'exception de la dernière.

  VB

  ```
  cmd.CommandText = _
      "SELECT * FROM Titles JOIN Publishers " _
      & "ON Publishers.PubId = Titles.PubID " _
      & "WHERE Publishers.State = 'CA'"
  ```

  L'utilisation de cette séquence simplifie la lecture de votre code, tant à l'écran que sur papier.

  Remarque

  Le caractère de continuation de ligne est nécessairement le dernier élément d'une ligne. Vous ne pouvez la faire suivre d'aucun autre élément sur cette même ligne.

  Il existe certaines restrictions relatives à l'emplacement du caractère de continuation de ligne ; ainsi, il est impossible de l'utiliser au milieu d'un nom d'argument. Vous pouvez couper une liste d'arguments à l'aide du caractère de continuation de ligne, mais les noms respectifs des arguments doivent rester intacts.

  Remarque

  Vous ne pouvez pas continuer un commentaire à l'aide d'un caractère de continuation de ligne. Une fois le commentaire commencé, le compilateur n'examine plus les caractères pour y rechercher une signification particulière. Dans le cas d'un commentaire à répartir sur plusieurs lignes, répétez le symbole de commentaire (') sur chaque ligne.

Même si la méthode recommandée consiste à ne faire figurer qu'une seule instruction par ligne, Visual Basic vous permet de placer plusieurs instructions sur une même ligne.

Pour placer plusieurs instructions sur une même ligne

- Séparez les instructions par un caractère deux-points (:), comme dans l'exemple suivant.

  VB

  ```
  text1.Text = "Hello" : text1.BackColor = System.Drawing.Color.Red
  ```

Voir aussi

Autres ressources

Structure de programme et conventions de codage

Comment : étiqueter des instructions

Les blocs d'instructions sont composés de lignes de code délimitées par le signe deux-points (:). Les lignes de code précédées d'une chaîne ou d'un entier les identifiant sont dites *étiquetées*. Les étiquettes d'instruction sont utilisées pour marquer une ligne de code afin de l'identifier pour être utilisée avec des instructions telles que **On Error Goto**.

Les étiquettes peuvent être soit des identificateurs Visual Basic 2005 valides, tels que ceux permettant d'identifier des éléments de programmation, soit des littéraux entiers. Une étiquette doit apparaître au début d'une ligne de code source et doit être suivie par le signe deux-points (:), qu'elle soit ou non suivie par une instruction sur la même ligne.

Le compilateur identifie les étiquettes en vérifiant si le début de la ligne correspond à un identificateur déjà défini. Si ce n'est pas le cas, le compilateur suppose que c'est une étiquette.

Les étiquettes possèdent leur propre espace de déclaration et n'interfèrent en rien avec d'autres identificateurs. La portée d'une étiquette est le corps de la méthode. La déclaration d'étiquette est prioritaire dans une situation ambiguë.

Remarque
Les étiquettes peuvent être utilisées uniquement sur des instructions exécutables à l'intérieur de méthodes.

Pour étiqueter une ligne de code.

- Placez un identificateur, suivi par le signe deux-points (:), au début de la ligne du code source.

  Par exemple, les lignes de code suivantes sont étiquetées respectivement par `Jump` et `120` :

  VB

```
Jump:   FileOpen(1, "testFile", OpenMode.Input)
        ' ...
120:    FileClose(1)
```

Voir aussi

Concepts

Vue d'ensemble des instructions

Autres ressources

Instructions dans Visual Basic

Structure de programme et conventions de codage

Caractères spéciaux dans le code

Vous êtes parfois amené à utiliser dans votre code des caractères spéciaux, c'est-à-dire des caractères qui ne figurent pas dans la liste des caractères alphanumériques standard. La ponctuation et les caractères spéciaux du jeu de caractères Visual Basic ont plusieurs utilisations, qui varient de l'organisation du texte de programme à la définition des tâches effectuées par le compilateur ou le programme compilé. Ils ne spécifient pas d'opération à effectuer.

Parenthèses

Vous utilisez des parenthèses lorsque vous définissez une procédure, par exemple un **Sub** ou **Function**. Vous devez placer les listes des arguments de la procédure entre parenthèses. Vous utilisez également des parenthèses pour insérer des variables ou des arguments dans des groupes logiques, en particulier pour substituer l'ordre de priorité des opérateurs par défaut dans une expression complexe. L'exemple suivant illustre ce comportement :

VB

```
Dim a, b, c, d, e As Double
a = 3.2
b = 7.6
c = 2
d = b + c / a
e = (b + c) / a
```

Après l'exécution du code précédent, `d` a la valeur 8.225 et `e` a la valeur 3. Le calcul de `d` utilise la priorité par défaut de **/** sur **+** et est équivalent à `d = b + (c / a)`. Les parenthèses dans le calcul de `e` substituent la priorité par défaut.

Séparateurs

Comme leur nom l'indique, les séparateurs séparent les différentes sections de code. En Visual Basic, c'est le signe deux-points (**:**) qui est le caractère séparateur. Les séparateurs vous permettent de placer plusieurs instructions sur une même ligne au lieu de lignes distinctes, ce qui constitue une économie d'espace et améliore la lisibilité de votre code. Dans l'exemple ci-dessous, trois instructions sont séparées par un signe deux-points (**:**).

VB

```
a = 3.2 : b = 7.6 : c = 2
```

Concaténation

Utilisez l'opérateur **&** pour créer une *concaténation*, c'est-à-dire lier plusieurs chaînes. Ne confondez pas cet opérateur avec l'opérateur **+**, qui permet d'ajouter des valeurs numériques. Si vous utilisez l'opérateur **+** pour créer une concaténation de valeurs numériques, vous risquez d'obtenir des résultats incorrects. C'est ce qu'illustre l'exemple suivant.

VB

```
var1 = "10.01"
var2 = 11
resultA = var1 + var2
resultB = var1 & var2
```

Après l'exécution du code précédent, `resultA` a la valeur 21.01 et `resultB` a la valeur 10.0111.

Opérateurs d'accès au membre

Pour accéder à un membre d'un type, utilisez le point (**.**) ou le point d'exclamation (**!**) comme opérateur entre le nom de type et le nom de membre.

Opérateur point (.)

Utilisez l'opérateur **.** sur une classe, une structure, une interface ou une énumération en tant qu'opérateur *d'accès au membre*. Le membre peut être un champ, une propriété, un événement, ou une méthode, comme l'illustre l'exemple ci-dessous.

VB

```
Dim nextForm As New System.Windows.Forms.Form
' Access Text member (property) of Form class (on nextForm object).
nextForm.Text = "This is the next form"
' Access Close member (method) on nextForm.
nextForm.Close()
```

Opérateur point d'exclamation (!)

Utilisez l'opérateur **!** uniquement dans une classe ou une interface en tant qu'opérateur *d'accès de type dictionnaire*. La classe ou l'interface doit avoir une propriété par défaut qui accepte un seul argument **String**. L'identificateur qui suit immédiatement l'opérateur **!** devient l'argument string pour la propriété par défaut. C'est ce qu'illustre l'exemple suivant.

VB

```
Public Class hasDefault
  Default Public ReadOnly Property index(ByVal s As String) As Integer
    Get
      Return 32768 + AscW(s)
    End Get
  End Property
End Class
Public Class testHasDefault
  Public Sub compareAccess()
    Dim hD As hasDefault = New hasDefault()
    MsgBox("Traditional access returns " & hD.index("X") & vbCrLf & _
      "Default property access returns " & hD("X") & vbCrLf & _
      "Dictionary access returns " & hD!X)
  End Sub
End Class
```

Toutes les trois lignes de sortie de **MsgBox** affichent la valeur `32856`. La première ligne utilise l'accès traditionnel à la propriété `index`, la seconde utilise le fait que `index` est la propriété par défaut de la classe `hasDefault` et la troisième utilise l'accès de type dictionnaire à la classe.

Notez que le second opérande de l'opérateur **!** doit être une chaîne qui n'est pas entourée par des guillemets doubles (**" "**). En d'autres termes, vous ne pouvez pas utiliser un littéral de chaîne ou une variable chaîne. La modification ci-dessous à la dernière ligne de l'appel **MsgBox** génère une erreur parce que `"X"` est un littéral de chaîne entouré par des guillemets.

```
"Dictionary access returns " & hD!"X")
```

Remarque

Les références aux collections par défaut doivent être explicites. En particulier, vous ne pouvez pas utiliser l'opérateur **!** dans une variable à liaison tardive.

Le signe **!** est également utilisé comme le caractère de type **Single**.

Voir aussi

Concepts

Caractères de type

Autres ressources

Structure de programme et conventions de codage

Commentaires dans le code

Lorsque vous lisez les exemples de code, vous rencontrez souvent le symbole de commentaire ('). Ce symbole indique au compilateur Visual Basic d'ignorer le texte qui le suit, c'est-à-dire le *commentaire*. Les commentaires sont de courtes explications destinées à éclairer ceux qui lisent le code.

En programmation, il est hautement recommandé de faire précéder toutes les procédures d'un commentaire court qui décrit les caractéristiques fonctionnelles de la procédure (ce qu'elle fait). Ceci est utile tant pour l'auteur du code lui-même que pour toute autre personne amenée à consulter le code. Vous devez séparer les détails relatifs à l'implémentation (comment la procédure fait ce qu'elle doit faire) des commentaires décrivant les caractéristiques fonctionnelles. Si vous fournissez des informations d'implémentation dans la description, n'oubliez pas de les mettre à jour lors de la mise à jour de la fonction.

Les commentaires peuvent soit suivre une instruction sur la même ligne, soit occuper une ligne entière. Ces deux cas sont illustrés par le code suivant :

VB

```
' This is a comment beginning at the left edge of the screen.
text1.Text = "Hi!"   ' This is an inline comment.
```

Si votre commentaire doit occuper plusieurs lignes, utilisez le symbole de commentaire sur chacune d'elles, comme l'exemple ci-dessous l'illustre.

VB

```
' This comment is too long to fit on a single line, so we break
' it into two lines. Some comments might need three or more lines.
```

Recommandations sur le commentaire

Le tableau suivant fournit des recommandations générales sur les types de commentaires qui peuvent précéder une section de code. Il ne s'agit que de suggestions ; Visual Basic n'impose aucune règle concernant l'ajout de commentaires. Procédez de la façon que vous jugez la plus efficace pour vous et toute personne amenée à lire votre code.

Type de commentaire	**Description du commentaire**
Purpose (but)	Décrit ce que fait la procédure (et non comment elle le fait)
Assumptions (Hypothèses)	Répertorie chaque variable externe, contrôle, fichier ouvert ou autre élément auquel la procédure accède.
Effects (Effets)	Répertorie chaque variable externe, contrôle ou fichier affecté, ainsi que ses effets (uniquement si ce n'est pas évident)
Inputs (Entrées)	Spécifie l'objectif attaché à l'argument
Returns (Retours)	Explique les valeurs retournées par la procédure.

N'oubliez pas :

- Pour chaque déclaration de variable importante, faites précéder d'un commentaire décrivant son utilisation.
- Les noms des variables, contrôles et procédures doivent être suffisamment clairs pour que les commentaires servent uniquement à fournir des détails d'implémentation complexes.
- Vous ne pouvez pas faire suivre une séquence de continuation par un commentaire sur la même ligne.

Vous pouvez ajouter ou supprimer les symboles de commentaires pour un bloc de code en sélectionnant au moins deux lignes de code et en choisissant les boutons **Commenter la sélection** ( ) et **Ne pas commenter la sélection** ( ) dans la barre

d'outils **Éditeur de texte**.

Remarque
Vous pouvez également ajouter des commentaires à votre code en faisant précéder le texte du mot clé **REM**. Cependant, le s ymbole **'** et les boutons **Commenter la sélection**/**Ne pas commenter la sélection** sont plus simples à utiliser et sont moi ns gourmands en espace et en mémoire.

Voir aussi

Référence

REM, instruction (Visual Basic)

Autres ressources

Structure de programme et conventions de codage

Comment : ajouter un commentaire à votre code (Visual Basic)

Cet exemple montre des commentaires occupant toute une ligne de code source, une partie de ligne et plusieurs lignes.

Exemple

VB

```
' This entire line is a comment.
Dim DailyTotal As Decimal = 0   ' Sales total for today.
' This comment is so long that it requires more than one line, so
' the comment character (') must be repeated on the second line.
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve dans **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite les éléments suivants :

- une référence à l'espace de noms System ;

Remarque

Vous ne pouvez pas continuer un commentaire à l'aide de la séquence de continuation de ligne (_). Vous devez répéter le caractère de commentaire (') au début de chaque ligne de commentaire.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Concepts

Commentaires dans le code

Documentation de votre code avec le langage XML (Visual Basic)

Dans Visual Basic, vous pouvez documenter votre code avec le langage XML.

Commentaires de documentation XML

Visual Basic fournit un moyen facile de créer automatiquement de la documentation XML pour les projets. Vous pouvez générer automatiquement une structure XML pour vos types et vos membres, puis fournir des résumés, une documentation descriptive pour chaque paramètre, ainsi que d'autres notes. Si la configuration est appropriée, la documentation XML est émise automatiquement dans un fichier XML portant le même nom que votre projet et une extension .xml. Pour plus d'informations, consultez /doc.

Le fichier XML peut être soit consommé, soit manipulé en tant que XML. Il se trouve dans le même répertoire que le fichier de sortie .exe ou .DLL de votre projet.

La documentation XML commence par `'''`. Le traitement de ces commentaires s'accompagne de certaines restrictions :

- Le XML utilisé dans la documentation doit être correct. Si le XML n'est pas correct, un avertissement est généré et le fichier de documentation contient un commentaire indiquant qu'une erreur est survenue.
- Les développeurs sont libres de créer leur propre jeu de balises. Il existe un jeu de balises recommandé (consultez la section « Rubriques connexes »). Certaines des balises recommandées ont des significations spéciales :
    - La balise <param> est employée pour décrire les paramètres. Si elle est utilisée, le compilateur vérifiera que le paramètre existe et que tous les paramètres sont décrits dans la documentation. Si la vérification échoue, il émet un avertissement.
    - L'attribut **cref** peut être rattaché à n'importe quelle balise pour fournir une référence à un élément de code. Le compilateur vérifie l'existence de cet élément de code. Si la vérification échoue, il émet un avertissement. Le compilateur respecte également les instructions **Imports** lorsqu'il recherche un type décrit dans l'attribut **cref**.
    - La balise <summary> est employée par IntelliSense dans Visual Studio pour afficher d'autres informations sur un type ou un membre.

Rubriques connexes

Pour plus d'informations sur la création d'un fichier XML avec des commentaires de documentation, consultez les rubriques suivantes :

- /doc
- Balises XML recommandées pour les commentaires de documentation (Visual Basic)
- Traitement du fichier XML (Visual Basic)
- Comment : créer une documentation XML en Visual Basic
- Commentaires XML, exemple
- Outils XML dans Visual Studio

Voir aussi

Autres ressources

Développement d'applications avec Visual Basic
Guide de programmation Visual Basic

Traitement du fichier XML (Visual Basic)

Le compilateur génère une chaîne d'identification pour chaque construction de votre code qui est placée entre balises de manière à générer de la documentation. (Pour plus d'informations sur la façon d'ajouter des balises à votre code, consultez Balises XML recommandées pour les commentaires de documentation (Visual Basic).) La chaîne d'identification identifie de manière unique la construction. Les programmes qui traitent le fichier XML peuvent utiliser la chaîne d'identification pour identifier l'élément de métadonnées/réflexion .NET Framework correspondant.

Le fichier XML n'est pas une représentation hiérarchique de votre code, mais une simple liste comportant un identificateur généré pour chaque élément.

Lorsqu'il génère les chaînes d'identification, le compilateur respecte les règles suivantes :

- La chaîne ne doit contenir aucun espace blanc.
- La première partie de la chaîne d'identification identifie le type de membre identifié, à l'aide d'un caractère unique suivi de deux points. Les types de membres utilisés sont présentés ci-dessous.

Caractère	Description
N	espace de noms Vous ne pouvez pas ajouter de commentaires de documentation à un espace de noms, mais vous pouvez ajouter des références CREF à ces commentaires, lorsqu'elles sont prises en charge.
T	type : **Class**, **Module**, **Interface**, **Structure**, **Enum**, **Delegate**
F	champ : **Dim**
P	propriété : **Property** (y compris les propriétés par défaut)
M	méthode : **Sub**, **Function**, **Declare**, **Operator**
E	événement : **Event**
!	chaîne d'erreur Le reste de la chaîne fournit des informations sur l'erreur. Le compilateur Visual Basic génère des informations d'erreur pour les liens qui ne peuvent pas être résolus.

- La deuxième partie de **String** est composée du nom complet de l'élément, à partir de la racine de l'espace de noms. Le nom de l'élément, ses types englobants et l'espace de noms sont séparés par des points. Si le nom de l'élément lui-même comporte des points, ils sont remplacés par un signe dièse (#). Par principe, aucun nom d'élément ne doit contenir de signe dièse à l'origine. Par exemple, le nom complet du constructeur **String** est `System.String.#ctor`.
- Pour les propriétés et méthodes, s'il existe des arguments pour la méthode, la liste des arguments figure à la suite entre parenthèses. En l'absence d'arguments, aucune parenthèse n'est mentionnée. Les arguments sont séparés par des virgules. Le codage de chaque argument correspond directement à son codage dans une signature .NET Framework.

Exemple

Le code ci-dessous illustre la manière dont les chaînes d'identification sont générées pour une classe et ses membres.

VB

```
Namespace SampleNamesapce

    ''' <summary>Signature is
    ''' "T:SampleNamesapce.SampleClass"
    ''' </summary>
    Public Class SampleClass
```

```vb
''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.#ctor"
''' </summary>
Public Sub New()
End Sub

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.#ctor(System.Int32)"
''' </summary>
Public Sub New(ByVal i As Integer)
End Sub

''' <summary>Signature is
''' "F:SampleNamesapce.SampleClass.SampleField"
''' </summary>
Public SampleField As String

''' <summary>Signature is
''' "F:SampleNamesapce.SampleClass.SampleConstant"
''' </summary>
Public Const SampleConstant As Integer = 42

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.SampleFunction"
''' </summary>
Public Function SampleFunction() As Integer
End Function

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.
''' SampleFunction(System.Int16[],System.Int32[0:,0:])"
''' </summary>
Public Function SampleFunction( _
    ByVal array1D() As Short, _
    ByVal array2D(,) As Integer) _
    As Integer
End Function

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.
''' op_Addition(SampleNamesapce.SampleClass,
'''             SampleNamesapce.SampleClass)"
''' </summary>
Public Shared Operator +( _
    ByVal operand1 As SampleClass, _
    ByVal operand2 As SampleClass) _
    As SampleClass

    Return Nothing
End Operator

''' <summary>Signature is
''' "P:SampleNamesapce.SampleClass.SampleProperty"
''' </summary>
Public Property SampleProperty() As Integer
    Get
    End Get
    Set(ByVal value As Integer)
    End Set
End Property

''' <summary>Signature is
''' "P:SampleNamesapce.SampleClass.Item(System.String)"
''' </summary>
Default Public ReadOnly Property Item( _
    ByVal s As String) As Integer

    Get
```

```
            End Get
        End Property

        ''' <summary>Signature is
        ''' "T:SampleNamesapce.SampleClass.NestedClass"
        ''' </summary>
        Public Class NestedClass
        End Class

        ''' <summary>Signature is
        ''' "E:SampleNamesapce.SampleClass.SampleEvent(System.Int32)"
        ''' </summary>
        Public Event SampleEvent As SampleDelegate

        ''' <summary>Signature is
        ''' "T:SampleNamesapce.SampleClass.SampleDelegate"
        ''' </summary>
        Public Delegate Sub SampleDelegate(ByVal i As Integer)
    End Class
End Namespace
```

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Référence

/doc

Comment : créer une documentation XML en Visual Basic

Cet exemple explique comment ajouter à votre code des commentaires de documentation XML.

Pour créer la documentation XML pour un type ou un membre

1. Dans l'**éditeur de code**, positionnez votre curseur sur la ligne au-dessus du type ou du membre pour lequel vous souhaitez créer la documentation.
2. Tapez `'''` (trois guillemets simples).

   Une structure XML pour le type ou le membre est ajoutée dans l'**éditeur de code**.
3. Ajoutez des informations descriptives entre les balises appropriées.

Remarque
Si vous ajoutez des lignes supplémentaires dans le bloc de documentation XML, chaque ligne doit commencer par `'''`.

4. Ajoutez du code supplémentaire qui utilise le type ou le membre avec les nouveaux commentaires de documentation XML.

   IntelliSense affiche le texte de la balise <summary> pour le type ou le membre.
5. Compilez le code pour générer un fichier XML qui contient les commentaires de documentation. Pour plus d'informations, consultez /doc.

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

/doc

Concepts

Documentation de votre code avec le langage XML (Visual Basic)

Restrictions liées à Visual Basic

Les versions précédentes de Visual Basic imposaient des limites pour le code, par exemple pour la longueur des noms de variables, le nombre de variables autorisé dans les modules et la taille des modules. Ces restrictions ont été levées dans Visual Basic 2005, ce qui vous laisse une plus grande liberté d'écriture et d'agencement pour votre code.

Les limites physiques dépendent plus de la mémoire au moment de l'exécution que des aspects pris en compte au moment de la compilation. Si vos pratiques en matière de programmation sont prudentes et que vous divisez les grandes applications en plusieurs classes et modules, les possibilités de rencontrer une limitation Visual Basic interne sont minimes.

Les éléments suivants constituent quelques limitations que vous pouvez rencontrer dans des cas extrêmes :

- **Longueur du nom.** Il existe un nombre maximal de caractères pour chaque élément de programmation déclaré. Ce maximum s'applique à une chaîne de qualification entière si le nom d'élément est qualifié. Consultez Noms d'éléments déclarés.
- **Longueur de ligne.** 65535 caractères au maximum peuvent être utilisés dans une ligne physique de code source. La ligne du code source logique peut être plus longue si vous utilisez des caractères de continuation de ligne. Consultez Comment : diviser et combiner des instructions dans le code.
- **Dimensions du tableau.** Il existe un nombre maximal de dimensions que vous pouvez déclarer pour un tableau. Cela limite le nombre d'index que vous pouvez utiliser pour spécifier un élément de tableau. Consultez Dimensions du tableau dans Visual Basic.
- **Longueur de chaîne.** Il existe un nombre maximal de caractères Unicode que vous pouvez stocker dans une chaîne unique. Consultez String, type de données (Visual Basic).
- **Longueur de chaîne d'environnement.** Il existe 32768 caractères au maximum pour toute chaîne d'environnement utilisée comme un argument de ligne de commande. Cette limitation existe sur toutes les plates-formes. Consultez Comment : accéder aux arguments de ligne de commande (Visual Basic).

Voir aussi

Concepts

Conventions d'affectation de noms Visual Basic

Autres ressources

Structure de programme et conventions de codage

Fonctionnalités du langage Visual Basic

Les rubriques suivantes présentent et décrivent les composants essentiels de Visual Basic, langage de programmation orienté objet. Après avoir créé l'interface de votre application à l'aide de formulaires et de contrôles, vous devez écrire le code qui définit le comportement de l'application. À l'instar de n'importe quel langage de programmation récent, Visual Basic prend en charge plusieurs constructions de programmation et éléments de langage classiques.

Si vous avez déjà utilisé d'autres langages de programmation, la plupart des concepts présentés dans cette section vous seront familiers. Alors que la plupart des structures sont les mêmes que celles d'autres langages, Visual Basic présente quelques différences subtiles dues au rôle déterminant que jouent les événements dans ce langage.

Si vous êtes novice en matière de programmation, les éléments présentés dans cette section constituent une introduction aux blocs de construction de base servant à écrire du code. Une fois les concepts fondamentaux assimilés, vous pouvez créer des applications puissantes avec Visual Basic.

Dans cette section

Visual Basic et le .NET Framework

Décrit le rôle de Visual Basic dans le .NET Framework et traite des assemblys et des attributs.

Éléments déclarés en Visual Basic

Décrit les éléments de programmation que vous pouvez déclarer, leurs noms et caractéristiques, ainsi que la manière dont le compilateur résout les références à ceux-ci.

Types de données en Visual Basic

Décrit les types de données que peut détenir un élément de programmation et le mode de stockage de ces données.

Variables en Visual Basic

Décrit comment déclarer des variables pour stocker des valeurs, ainsi que la manière de les utiliser pour faire référence à des objets.

Tableaux dans Visual Basic

Explique comment rendre votre code plus condensé et puissant en déclarant et en utilisant des tableaux qui contiennent plusieurs valeurs connexes.

Objets en Visual Basic

Propose une vue d'ensemble des objets et des classes, de leur mode d'utilisation, de leurs relations entre eux ainsi que des propriétés, des méthodes et des événements qu'ils exposent.

Chaînes en Visual Basic

Aborde la déclaration et la manipulation de variables qui contiennent un ensemble de caractères.

Constantes et énumérations en Visual Basic

Décrit le stockage de valeurs inchangées pour un usage fréquent, notamment de jeux de valeurs constantes connexes.

Opérateurs et expressions en Visual Basic

Décrit les éléments de code qui manipulent des éléments contenant des valeurs, ainsi que la manière de les utiliser efficacement et de les combiner pour obtenir de nouvelles valeurs.

Instructions dans Visual Basic

Offre une vue d'ensemble des deux principaux types d'instructions dans Visual Basic : les instructions de déclaration et les instructions exécutables.

Procédures dans Visual Basic

Décrit les procédures **Sub**, **Function**, **Property** et **Operator**, ainsi que des sujets plus avancés, tels que les procédures récurrentes et surchargées.

Flux de contrôle en Visual Basic

Décrit comment contrôler le déroulement de l'exécution d'un programme.

Rubriques connexes

Programmation orientée objet dans Visual Basic

Traite de la programmation orientée objet, notamment des classes et interfaces, de la création d'objets, des événements et délégués, de l'héritage ainsi que des groupes d'objets.

Référence Visual Basic

Fournit des informations de référence sur les différents aspects de la programmation Visual Basic.

Visual Basic et le .NET Framework

Visual Basic s'appuie sur le .NET Framework qui fournit une sécurité améliorée, la gestion de la mémoire, le versioning ainsi que la prise en charge du déploiement. Le .NET Framework offre également une interopérabilité entre les objets que vous créez à partir des langages de programmation .NET Framework. Cela signifie que vous pouvez créer des objets avec Visual Basic qui sont faciles à utiliser à partir d'autres langages du .NET Framework, et que vous pouvez utiliser les objets des autres langages du .NET Framework de la même façon que les objets créés avec Visual Basic.

Dans cette section

Assemblys

Explique ce que sont les assemblys et comment les utiliser.

Références et l'instruction Imports

Explique comment référencer les objets dans d'autres assemblys

Espaces de noms dans Visual Basic

Explique comment les espaces de noms organisent les objets dans les assemblys.

Attributs en Visual Basic

Explique comment les attributs .NET Framework fournissent des informations supplémentaires sur les entités définies à l'intérieur d'une application Visual Basic.

Rubriques connexes

Comment : ajouter ou supprimer des références en Visual Studio

Explique le processus d'ajout de références afin de pouvoir écrire du code pour des composants externes, et explique comment supprimer les références inutiles.

Vue d'ensemble conceptuelle du .NET Framework

Présente l'architecture du .NET Framework et ses composants.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Présente une liste et une description des éléments de langage ayant été modifiés.

Assemblys

Les assemblys constituent l'unité fondamentale de déploiement, le contrôle des versions, la réutilisation, la portée d'activation et les autorisations de sécurité d'une application .NET. Les assemblys se présentent sous la forme d'un fichier exécutable (.exe) ou d'un fichier de bibliothèque de liens dynamiques (.dll), et constituent les blocs de construction du .NET Framework. Ils fournissent le Common Language Runtime ainsi que les informations nécessaires à la reconnaissance des implémentations des types. Un assembly ressemble à une collection de types et de ressources qui forment une unité de fonctionnalité logique et sont conçus pour fonctionner ensemble.

Grâce à Visual Basic, vous pouvez utiliser le contenu des assemblys et leur ajouter des références, de la même façon que les bibliothèques de types des précédentes versions de Visual Basic. Cependant, la différence entre les assemblys et les fichiers .exe ou .dll des précédentes versions de Windows réside dans le fait qu'ils contiennent toutes les informations disponibles dans une bibliothèque de types, ainsi que toutes les informations nécessaires à l'utilisation de l'application ou du composant.

Manifeste d'assembly

Tous les assemblys contiennent un *manifeste d'assembly*. Semblable à un sommaire, le manifeste d'assembly contient les éléments suivants :

- L'identité de l'assembly (nom et version).
- Table de fichiers décrivant tous les autres fichiers qui constituent l'assembly, par exemple, tous les autres assemblys créés pour vos fichiers .exe ou .dll, voire des fichiers bitmap ou Readme.
- *Liste de référence de l'assembly*, qui est une liste de toutes les dépendances externes (fichiers .dll ou autres nécessaires à votre application, mais qui ont peut-être été créés par une autre personne). Les références de l'assembly contiennent des références aux objets globaux et privés. Les objets globaux résident dans le Global Assembly Cache, une zone disponible pour les applications tierces, qui est comparable au répertoire System32. L'espace de noms Microsoft.VisualBasic est un exemple d'assembly dans le Global Assembly Cache. Les objets privés doivent se trouver dans un répertoire situé au même niveau ou à un niveau inférieur au répertoire d'installation de votre application.

Étant donné que les assemblys contiennent des informations sur le contenu, le versioning et les dépendances, les applications créées à l'aide de Visual Basic ne s'appuient pas sur les valeurs de Registre pour fonctionner correctement. Les assemblys permettent de réduire les conflits de DLL et rendent vos applications plus fiables, plus faciles à déployer. Dans la plupart des cas, vous pouvez installer une application .NET en copiant simplement ses fichiers sur l'ordinateur cible.

Références

Pour utiliser un assembly, vous devez ajouter une référence à celui-ci, comme décrit dans Comment : ajouter ou supprimer des références en Visual Studio. Ensuite, vous devez utiliser l'instruction **Imports** pour choisir l'espace de noms des éléments que vous souhaitez utiliser, comme décrit dans la rubrique Références et l'instruction Imports. Une fois qu'un assembly est référencé et importé, toutes les classes, les propriétés, les méthodes et les autres membres accessibles de ses espaces de noms sont disponibles dans votre application comme si leur code faisait partie de votre fichier source. Un seul assembly peut contenir plusieurs espaces de noms, et chaque espace de noms peut contenir différents regroupements d'éléments, y compris d'autres espaces de noms.

Pour plus d'informations sur la création d'assemblys, consultez Comment : créer et utiliser des assemblys.

Voir aussi

Tâches

Comment : créer et utiliser des assemblys
Comment : ajouter ou supprimer des références en Visual Studio
Comment : afficher le contenu d'un assembly

Référence

Imports, instruction
Microsoft.VisualBasic

Concepts

Espaces de noms dans Visual Basic
Références et l'instruction Imports

Autres ressources

Assemblys dans le Common Language Runtime
Modifications du langage pour les utilisateurs de Visual Basic 6.0

Comment : créer et utiliser des assemblys

Un assembly se compose d'un ou plusieurs fichiers .exe ou .dll qui constituent une application Visual Studio. Les assemblys sont créés automatiquement lorsque vous compilez des fichiers sources Visual Basic.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour créer un assembly

- Compilez votre application en cliquant sur **Générer** dans le menu **Générer** ou en générant celle-ci à partir de la ligne de commande via le compilateur de ligne de commande. Pour plus d'informations sur la génération d'assemblys à partir de la ligne de commande, consultez Génération à partir de la ligne de commande (Visual Basic).

Pour ajouter une référence à un autre assembly

- Cliquez sur la commande **Ajouter une référence** dans le menu **Projet**, puis sélectionnez l'assembly que vous souhaitez utiliser. Pour plus d'informations, consultez Comment : ajouter ou supprimer des références en Visual Studio.

Pour utiliser des objets dans un autre assembly

- Spécifiez le nom complet de l'objet ou utilisez un alias ayant été défini pour cet objet à l'aide d'une instruction **Imports**. Pour plus d'informations sur les noms qualifiés complets, consultez Espaces de noms dans Visual Basic. Pour plus d'informations sur l'ajout de références et l'utilisation de l'instruction **Imports**, consultez Références et l'instruction Imports.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio
Comment : afficher le contenu d'un assembly

Référence

Imports, instruction

Concepts

Assemblys
Création d'assemblys
Espaces de noms dans Visual Basic
Références et l'instruction Imports

Autres ressources

Assemblys dans le Common Language Runtime
Modifications du langage pour les utilisateurs de Visual Basic 6.0
Génération à partir de la ligne de commande (Visual Basic)

Références et l'instruction Imports

Vous pouvez rendre les objets externes disponibles pour votre projet en cliquant sur la commande **Ajouter une référence** dans le menu **Projet**. Les références en Visual Basic peuvent pointer vers des assemblys qui ressemblent à des bibliothèques de types, mais contiennent plus d'informations.

L'instruction Imports

Les assemblys incluent un ou plusieurs espaces de noms. Lorsque vous ajoutez une référence à un assembly, vous pouvez également ajouter une instruction **Imports**à un module pour contrôler la visibilité des espaces de noms de l'assembly dans le module. L'instruction **Imports** offre une portée qui vous permet d'utiliser uniquement la portion d'espace de noms nécessaire pour fournir une référence unique.

La syntaxe de l'instruction **Imports** est la suivante :

Imports [**|** *Aliasname* =] *Namespace*

Aliasname fait référence à un nom court que vous pouvez utiliser dans code pour faire référence à un espace de noms importé. *Namespace* est un espace de noms disponible par le biais d'une référence de projet, d'une définition dans le projet ou d'une instruction **Imports** antérieure.

Un module peut contenir autant d'instructions **Imports** que vous le souhaitez. Celles-ci doivent apparaître après toutes les instructions **Option**, le cas échéant, mais avant tout autre code.

Remarque

Ne confondez pas les références de projet avec les instructions **Imports** ou **Declare**. Les références de projet rendent les obj ets externes, tels que les objets des assemblys, disponibles pour les projets Visual Basic. L'utilisation de l'instruction **Imports** permet de simplifier l'accès aux références de projet, mais ne constitue pas un accès à ces objets. L'instruction **Declare** perm et de déclarer une référence à une procédure externe dans une bibliothèque de liens dynamiques (DLL).

Utilisation des alias avec l'instruction Imports

L'instruction **Imports** facilite l'accès aux méthodes des classes en éliminant le besoin d'entrer de manière explicite les noms complets des références. Les alias vous permettent d'assigner un nom plus convivial à seulement une partie d'un espace de noms. Par exemple, la séquence du retour chariot ou du changement de ligne qui entraîne l'affichage d'un bloc de texte sur plusieurs lignes fait partie du module ControlChars, module (Visual Basic) de l'espace de noms Microsoft.VisualBasic. Pour utiliser cette constante dans un programme sans recourir à un alias, vous devez entrer le code suivant :

VB

```
MsgBox("Some text" & Microsoft.VisualBasic.ControlChars.CrLf _
    & "Some more text")
```

Les instructions **Imports** doivent toujours représenter les premières lignes situées immédiatement après les instructions **Option** d'un module. Le fragment de code suivant montre comment importer et assigner un alias au module Microsoft.VisualBasic.ControlChars:

VB

```
Imports CtrlChrs = Microsoft.VisualBasic.ControlChars
```

Les références ultérieures à cet espace de noms peuvent être considérablement plus courtes :

VB

```
MsgBox("Some text" & CtrlChrs.CrLf & "Some more text")
```

Si une instruction **Imports** n'inclut pas de nom d'alias, les éléments définis dans l'espace de noms importé peuvent être utilisés dans le module sans nécessiter de qualification. Si le nom d'alias est spécifié, il doit être utilisé comme qualificateur

pour les noms contenus dans l'espace de noms.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Comment : créer et utiliser des assemblys

Référence

Imports, instruction

ControlChars, module (Visual Basic)

Microsoft.VisualBasic

Concepts

Espaces de noms dans Visual Basic

Assemblys

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Espaces de noms dans Visual Basic

Les espaces de noms permettent d'organiser les objets définis dans un assembly. Les assemblys peuvent contenir plusieurs espaces de noms, qui peuvent à leur tour contenir d'autres espaces de noms. Les espaces de noms permettent d'éviter les ambiguïtés et de simplifier les références lors de l'utilisation de grands groupes d'objets, tels que les bibliothèques de classes.

Par exemple, le .NET Framework définit la classe ListBox dans l'espace de noms System.Windows.Forms. Le fragment de code suivant montre comment déclarer une variable en utilisant le nom complet de cette classe :

VB

```
Dim LBox As System.Windows.Forms.ListBox
```

Comment éviter les collisions de noms

Les espaces de noms du .NET Framework permettent de résoudre le problème de l'*encombrement des espaces de noms*, par exemple lorsque le développeur d'une bibliothèque de classes est gêné par l'utilisation de noms similaires dans une autre bibliothèque. Ces conflits avec des composants existants sont parfois appelés *collisions de noms*.

Par exemple, si vous créez une nouvelle classe nommée `ListBox`, vous pouvez l'utiliser dans votre projet sans qualification. Cependant, si vous souhaitez utiliser la classe **ListBox** du .NET Framework dans le même projet, vous devez utiliser un nom complet pour rendre la référence unique. Si la référence n'est pas unique, Visual Basic génère une erreur en indiquant que le nom est ambigu. L'exemple de code suivant montre comment déclarer ces objets :

VB

```
' Define a new object based on your ListBox class.
Dim LBC As New ListBox
' Define a new Windows.Forms ListBox control.
Dim MyLB As New System.Windows.Forms.ListBox
```

L'illustration suivante montre deux hiérarchies d'espaces de noms, chacune contenant un objet nommé `ListBox`.

Par défaut, tous les fichiers exécutables créés à l'aide de Visual Basic contiennent un espace de noms portant le même nom que votre projet. Par exemple, si vous définissez un objet dans un projet nommé `ListBoxProject`, le fichier exécutable, ListBoxProject.exe, contient un espace de noms appelé `ListBoxProject`.

Plusieurs assemblys peuvent utiliser le même espace de noms. Visual Basic les traite comme un seul ensemble de noms. Par exemple, vous pouvez définir des classes pour un espace de noms appelé `SomeNameSpace` dans un assembly nommé `Assemb1`, et définir des classes supplémentaires pour le même espace de noms à partir d'un assembly nommé `Assemb2`.

Noms complets

Les noms complets sont des références d'objet qui sont préfixées à l'aide du nom de l'espace de noms dans lequel l'objet est défini. Vous pouvez utiliser les objets définis dans d'autres projets si vous créez une référence à la classe (en cliquant sur **Ajouter une référence** dans le menu **Projet**) et si vous utilisez le nom complet de l'objet dans votre code. Le fragment de code suivant montre comment utiliser le nom complet d'un objet provenant de l'espace de noms d'un autre projet :

VB

```
Dim LBC As New ListBoxProject.Form1.ListBox
```

Les noms complets empêchent les conflits d'attribution de noms parce qu'ils permettent au compilateur de déterminer l'objet utilisé. Cependant, ces noms peuvent devenir longs et encombrants. Pour contourner ce problème, utilisez l'instruction **Imports** pour définir un *alias* (nom abrégé que vous pouvez utiliser à la place d'un nom complet). Par exemple, l'exemple de code suivant crée les alias de deux noms complets, puis utilise ces alias pour définir deux objets.

VB

```
Imports LBControl = System.Windows.Forms.ListBox
Imports MyListBox = ListBoxProject.Form1.ListBox
```

VB

```
Dim LBC As LBControl
Dim MyLB As MyListBox
```

Si vous utilisez l'instruction **Imports** sans alias, vous pouvez vous servir de tous les noms de l'espace de noms sans qualification, à condition qu'ils soient uniques à l'intérieur du projet. Si votre projet contient des instructions **Imports** pour les espaces de noms contenant des éléments qui portent le même nom, vous devez fournir un nom complet lorsque vous l'utilisez. Supposons, par exemple, que votre projet contienne les deux instructions **Imports** suivantes :

VB

```
' This namespace contains a class called Class1.
Imports MyProj1
' This namespace also contains a class called Class1.
Imports MyProj2
```

Si vous essayez d'utiliser `Class1` sans fournir son nom complet, Visual Basic génère une erreur en indiquant que le nom `Class1` est ambigu.

Instructions au niveau de l'espace de noms

Dans un espace de noms, vous pouvez définir des éléments tels que des modules, des interfaces, des classes, des délégués, des énumérations, des structures et d'autres espaces de noms. Vous ne pouvez pas définir les éléments tels que les propriétés, les procédures, les variables et les événements au niveau de l'espace de noms. Ces éléments doivent être déclarés dans des conteneurs tels que les modules, les structures ou les classes.

Remarque

Si vous avez défini une hiérarchie imbriquée d'espaces de noms, le code à l'intérieur de cette hiérarchie peut ne pas pouvoir accéder aux classes à l'intérieur d'autres espaces de noms ayant le même nom. Par exemple, si vous définissez un espace de noms nommé `System` dans un espace de noms nommé `SpecialSpace`, les membres de l'espace de noms **System** du .NET Fr amework ne seront pas accessibles à moins que vous ne qualifiiez complètement les membres avec le mot clé **Global**. Pour plus d'informations, consultez Global.

Voir aussi

Tâches

Comment : créer et utiliser des assemblys

Référence

Global
Imports, instruction
ListBox
System.Windows.Forms

Concepts

Assemblys
Références et l'instruction Imports

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Attributs en Visual Basic

Les attributs sont des balises semblables aux mots clés dans lesquelles il est possible de spécifier des informations supplémentaires au sujet des entités définies dans les applications Visual Basic. Enregistrés avec les métadonnées d'un assembly, les attributs annotent des éléments de programmation tels que les types, les champs, les méthodes et les propriétés. D'autres applications, telles que les compilateurs Visual Basic ou ASP.NET, peuvent se référer aux informations figurant dans les attributs pour déterminer comment les objets doivent être utilisés.

Les sections suivantes fournissent des informations supplémentaires au sujet des attributs et de la manière de les définir, de les appliquer et de les récupérer.

Dans cette section

Vue d'ensemble des attributs dans Visual Basic

Fournit des informations sur les attributs ainsi que sur leurs fonctions et fonctionnalités.

Attributs utilisés dans Visual Basic

Répertorie certains des attributs que vous allez très probablement utiliser en Visual Basic.

Utilisations courantes des attributs

Décrit certaines des utilisations les plus fréquentes des attributs.

Application des attributs

Traite de l'application des attributs aux éléments de programme.

Attributs globaux en Visual Basic

Propose des informations sur les attributs s'appliquant aux modules ou aux assemblys.

Attributs personnalisés en Visual Basic

Explique comment définir vos propres attributs.

Rubriques connexes

Programmation orientée objet dans Visual Basic

Fournit des informations au sujet de la programmation orientée objet et sur son utilisation.

Métadonnées et composants autodescriptifs

Fournit des informations détaillées sur les types de métadonnées utilisés dans Visual Studio, y compris sur les attributs.

Vue d'ensemble des attributs dans Visual Basic

Les attributs sont des balises descriptives qui fournissent des informations supplémentaires sur des éléments de programmation, tels que les types, les champs, les méthodes et les propriétés. D'autres applications, telles que le compilateur Visual Basic, peuvent se référer aux informations complémentaires figurant dans les attributs pour déterminer la manière dont il faut utiliser ces éléments.

Attributs et métadonnées

Les attributs sont enregistrés avec les *métadonnées* d'assemblys Visual Basic. Les métadonnées sont des informations qui décrivent tous les éléments managés par le runtime. Il peut s'agir notamment d'informations requises pour les opérations de débogage et garbage collection, ainsi que des attributs de sécurité, des données de marshaling, des définitions de membres et de classes étendues, de la liaison des versions et de toute autre information indispensable au runtime.

Avec les attributs, vous devez spécifier les métadonnées pratiquement de la même façon que vous utilisez des mots clés comme **Public** et **Private** pour fournir des informations sur les niveaux d'accès. Toutefois, à la différence des mots clés, la plupart des attributs ne sont pas spécifiques au langage. À l'aide d'attributs, vous pouvez étendre les fonctionnalités du langage Visual Basic sans que cela ne nécessite l'apport de modifications au compilateur.

Le .NET Framework et le langage Visual Basic définissent un grand nombre d'attributs utiles, et vous pouvez définir vos propres attributs personnalisés qui sont significatifs pour votre application. Les attributs personnalisés sont basés sur la classe System.Attribute, et ils utilisent l'attribut AttributeUsageAttribute pour fournir des informations supplémentaires sur la manière d'utiliser l'attribut.

Fonction et fonctionnalités des attributs

Voici quelques-uns des principaux points à retenir au sujet des attributs :

- Vous pouvez appliquer un ou plusieurs attributs à des modules ou à des assemblys tout entiers, ou à des éléments de programme plus petits, comme les classes et les propriétés.
- Les attributs peuvent accepter des arguments, de la même façon que les méthodes et les propriétés.
- Le processus qui consiste à extraire des métadonnées des attributs se nomme *réflexion*. La réflexion implique l'emploi d'outils permettant aux objets d'extraire et d'examiner des métadonnées relatives à leurs propres membres. Pour plus d'informations, consultez Réflexion, exemple.

Voir aussi

Tâches

Comment : définir vos propres attributs

Référence

AttributeUsageAttribute

System.Attribute

Concepts

Utilisations courantes des attributs

Application des attributs

Attributs globaux en Visual Basic

Attributs utilisés dans Visual Basic

Attributs et prise en charge au moment du design

Autres ressources

Attributs personnalisés en Visual Basic

Attributs utilisés dans Visual Basic

Le tableau suivant répertorie certains des attributs les plus courants utilisés dans Visual Basic. Pour obtenir plus d'informations sur un attribut spécifique, tapez son nom dans la zone Index ou Recherche du système d'aide.

Attribut	But
ComClassAttribute, classe	Indique au compilateur que la classe doit être exposée comme un objet COM. Propre à Visu al Basic.
HideModuleNameAttribute, classe	Permet aux membres de module d'être accessibles à l'aide uniquement de la qualification n écessaire pour le module.
VBFixedStringAttribute, classe	Spécifie la taille d'une chaîne de longueur fixe dans une structure à utiliser avec les fonction s d'entrée et de sortie de fichier. Propre à Visual Basic.
VBFixedArrayAttribute, classe	Spécifie la taille d'un tableau fixe dans une structure à utiliser avec les fonctions d'entrée et de sortie de fichier. Propre à Visual Basic.
WebMethodAttribute	Autorise l'appel d'une méthode à l'aide du protocole SOAP. Utilisé dans les services Web X ML.
SerializableAttribute	Indique qu'une classe peut être sérialisée.
MarshalAsAttribute	Détermine comment un paramètre doit être marshalé entre le code managé de Visual Basic et le code non managé (d'une API Windows par exemple). Utilisé par le Common Language Runtime.
AttributeUsageAttribute	Spécifie l'usage qui peut être fait d'un attribut.
DllImportAttribute	Indique que la méthode attribuée est implémentée comme une exportation à partir d'une D LL non managée.

Attributs propres à Visual Basic

Trois attributs sont spécifiques à Visual Basic : **COMClassAttribute**, **VBFixedStringAttribute** et **VBFixedArray**.

COMClassAttribute

Utilisez l'attribut **COMClassAttribute** pour simplifier le processus de création de composants COM à partir de Visual Basic. Les objets COM sont très différents des assemblys .NET Framework et, sans l'attribut **COMClassAttribute**, vous devez mettre en œuvre un certain nombre d'opérations pour générer un objet COM à partir de Visual Basic. Nombre de ces opérations sont automatisées par le compilateur lorsqu'il rencontre des classes marquées avec l'attribut **COMClassAttribute**.

HideModuleNameAttribute

Utilisez **HideModuleNameAttribute** pour permettre aux membres de module d'être accessibles à l'aide uniquement de la qualification nécessaire pour le module.

VBFixedStringAttribute

Utilisez **VBFixedStringAttribute** pour forcer Visual Basic à créer une chaîne de longueur fixe. Les chaînes ont une taille variable par défaut, et cet attribut est utile pour stocker des chaînes dans des fichiers. Le code suivant illustre cette méthode :

VB

```
Structure Worker
    ' The runtime uses VBFixedString to determine
    ' if the field should be written out as a fixed size.
    <VBFixedString(10)> Public LastName As String
    <VBFixedString(7)> Public Title As String
    <VBFixedString(2)> Public Rank As String
```

```
End Structure
```

VBFixedArrayAttribute

L'attribut **VBFixedArrayAttribute** permet de déclarer des tableaux de taille fixe. À l'instar des chaînes Visual Basic, les tableaux ont une longueur variable par défaut. Cet attribut est utile lors de la sérialisation ou de l'écriture de données dans des fichiers.

Voir aussi

Référence

ComClassAttribute, classe
VBFixedArrayAttribute, classe
VBFixedStringAttribute, classe
System.Runtime.InteropServices

Concepts

Attributs globaux en Visual Basic
Application des attributs
Récupération des informations stockées dans les attributs
Écriture des attributs personnalisés

Autres ressources

Extension des métadonnées à l'aide des attributs

Utilisations courantes des attributs

L'utilisation du Common Language Runtime et des bibliothèques de classes vous amènera probablement à devoir employer des attributs à un moment donné. La liste suivante décrit certaines des utilisations courantes des attributs dans le code :

- Marquer une méthode à l'aide de l'attribut **WebMethod** des services Web XML pour indiquer que la méthode doit pouvoir être appelée à l'aide du protocole SOAP. Pour plus d'informations, consultez WebMethodAttribute.
- Décrire la manière dont il faut marshaler les paramètres de méthode en cas d'interopérabilité avec du code natif Pour plus d'informations, consultez MarshalAsAttribute.
- Décrire des propriétés COM de classes, méthodes et interfaces
- Marquer des composants comme des composants COM pour que le compilateur Visual Basic génère le code supplémentaire requis pour créer un tel composant Pour plus d'informations, consultez ComClassAttribute.
- Appeler du code non managé à l'aide de la classe DllImportAttribute.
- Décrire un assembly au moyen d'un titre, d'une version, d'une description ou d'une marque
- Désigner des membres d'une classe à sérialiser en vue de leur persistance
- Décrire comment mapper des membres de classe et des nœuds XML dans le cadre de la sérialisation XML
- Définir les impératifs en matière de sécurité des méthodes
- Spécifier les caractéristiques employées pour appliquer la sécurité
- Contrôler les optimisations à l'aide du compilateur JIT (Just-In-Time) pour que le code demeure facile à déboguer

Vous pouvez utiliser les attributs de nombreuses autres manières et vous pouvez également créer des attributs personnalisés.

Voir aussi

Tâches

Comment : définir vos propres attributs

Concepts

Attributs et prise en charge au moment du design
Vue d'ensemble des attributs dans Visual Basic
Application des attributs
Attributs globaux en Visual Basic
Attributs utilisés dans Visual Basic

Autres ressources

Attributs personnalisés en Visual Basic

Application des attributs

L'application d'attributs s'effectue par l'ajout de blocs d'attributs à des éléments de programme, tels que propriétés, méthodes, événements, classes et assemblys. Un bloc d'attributs se compose d'une liste de déclarations attribute séparées par des virgules et encadrée par des crochets pointus (< >). Une déclaration d'attribut consiste en un modificateur d'attribut facultatif, comme **Module** ou **Assembly**, en un nom d'attribut, en une liste (éventuellement surchargée) de paramètres positionnels requis et en une liste d'arguments nommés facultatifs. Vous devez placer les attributs avec des modificateurs dans une section d'attributs en haut d'un fichier source. Par exemple, le code suivant définit un attribut d'assembly qui détermine un titre pour l'assembly et un attribut de module indiquant que le module est conforme CLS (Common Language Specification) :

VB

```
Imports System.Reflection
<Assembly: AssemblyTitleAttribute("Production assembly 4"), _
Module: CLSCompliant(True)>
```

Les attributs d'assembly peuvent également être appliqués à l'aide du fichier AssemblyInfo.vb, qui est automatiquement ajouté aux projets par l'interface utilisateur de Visual Studio. Ce fichier contient des attributs d'assembly dotés soit de valeurs par défaut, soit de valeurs vides.

Lorsqu'ils sont appliqués aux éléments de programme (par exemple, aux propriétés), les attributs précèdent l'élément. Par exemple, le code suivant applique un attribut à une définition de classe :

VB

```
<CustomAttr(Update:=True)> Class Class1
```

Par convention, tous les noms d'attributs se terminent par le mot "Attribute" de façon à les distinguer des autres éléments du .NET Framework. Cependant, vous n'êtes pas obligé de spécifier le suffixe d'attribut lorsque vous employez des attributs. Prenons par exemple un attribut intitulé `CustomAttrAttribute` : `<CustomAttr(Update:=True)>` est l'équivalent de `<CustomAttrAttribute(Update:=True)>`.

Arguments d'attribut

À la manière des objets, les attributs utilisent des arguments optionnels, obligatoires, positionnels et nommés. Les arguments positionnels sont des arguments qui sont spécifiés dans l'ordre dans lequel ils sont déclarés pour le constructeur d'un attribut. Par exemple, le code suivant appelle le constructeur **Sub New** d'un attribut possédant deux valeurs :

VB

```
<CustomAttr(True, False)> Class Class1
```

Les arguments passés à **Sub New** dans les classes d'attributs sont souvent employés pour initialiser les valeurs des champs et propriétés.

Vous pouvez recourir à des arguments nommés pour définir directement la valeur des propriétés et champs. Pour spécifier un argument nommé, placez ":=" à la fin du nom de l'argument, suivi de la valeur définie. Contrairement aux arguments positionnels, les arguments nommés peuvent être spécifiés dans n'importe quel ordre. Par exemple, le code suivant attribue la valeur **True** au champ `Update` et la valeur **False** au champ `Keep` :

VB

```
<CustomAttr(Update:=True, Keep:=False)> Class Class1
```

Remarque

Les arguments d'attributs présentent une différence importante par rapport aux arguments utilisés avec les appels de méthode standard. Vous devez avoir recours à des arguments positionnels employés avec le constructeur **Sub New** de classes d'attributs. Les arguments nommés ne peuvent être utilisés que pour définir les valeurs des zones et des propriétés des classes d'attributs.

Les arguments obligatoires sont ceux qui doivent toujours être spécifiés. Quant aux arguments optionnels, il s'agit d'arguments qui peuvent être omis, soit par l'emploi d'une virgule comme espace réservé lors de l'utilisation d'arguments positionnels, soit par leur absence pure et simple lors de l'utilisation d'arguments nommés.

Les arguments d'attributs doivent être des expressions constantes.

Exemples d'attributs

Les procédures suivantes contiennent des exemples de déclarations attribute :

Pour utiliser l'attribut MarshalAs afin de contrôler la manière dont les paramètres sont marshalés

1. Ajoutez une instruction **Imports** à l'espace de noms System.Runtime.InteropServices en haut de votre code source :

   VB

   ```
   Imports System.Runtime.InteropServices
   ```

2. Faites précéder les paramètres de l'attribut MarshalAsAttribute et spécifiez le type de données requis par la cible. Par exemple, le code suivant marshale deux paramètres comme type de données de pointeur long en string (**LPStr**) pour une fonction API Windows :

   VB

   ```
   Declare Auto Sub CopyFile Lib "Kernel32.Lib" ( _
      <MarshalAs(UnmanagedType.LPStr)> ByVal existingfile As String, _
      <MarshalAs(UnmanagedType.LPStr)> ByVal newfile As String, _
      ByVal failifexists As Boolean _
   )
   ```

Le Common Language Runtime utilise l'attribut **MarshalAsAttribute** pour déterminer comment un paramètre doit être marshalé entre le code managé de Visual Basic et le code non managé d'un appel API Windows.

Pour exposer une méthode à des clients Web distants

1. Dans le menu **Fichier**, cliquez sur **Projet**, sélectionnez le modèle **ASP.NET Web Service** et ajoutez une instruction **Imports** pour l'espace de noms System.Web :

   VB

   ```
   Imports System.Web.Services
   ```

2. Définissez une méthode et, à l'aide de l'attribut WebMethodAttribute, faites en sorte qu'elle puisse être appelée à partir de clients Web distants :

   VB

   ```
   <WebMethod()> Public Function HelloWorld() As String
       HelloWorld = "Hello World..."
   End Function
   ```

Il ne suffit pas de rendre **Public** une méthode d'un service Web XML pour l'exposer aux clients Web. L'attribut

WebMethodAttribute doit être appliqué explicitement à une méthode pour que cette dernière puisse être appelée par un client Web distant.

Voir aussi

Tâches

Comment : définir vos propres attributs

Référence

Imports, instruction

WebMethodAttribute

MarshalAsAttribute

System.Web

Concepts

Vue d'ensemble des attributs dans Visual Basic

Utilisations courantes des attributs

Passage des arguments par position et par nom

Autres ressources

Attributs personnalisés en Visual Basic

Attributs globaux en Visual Basic

La plupart des attributs sont attachés à des éléments spécifiques du langage, comme les classes ou les méthodes. Toutefois, il existe également des attributs dits globaux, qui peuvent s'appliquer à un assembly ou à un module tout entier.

Vous pouvez définir la plupart des attributs dans l'environnement de développement intégré (IDE) Visual Studio à l'aide de la Informations de l'assembly, boîte de dialogue. Pour plus d'informations, consultez Gestion des propriétés de l'application et Gestion d'assembly et signature de manifeste.

Attributs d'assembly

La syntaxe suivante permet de spécifier des attributs au niveau assembly :

```
<Assembly: Attribute1, Assembly: Attribute2..., Assembly: AttributeN>
```

Une syntaxe similaire permet de spécifier des attributs au niveau module :

```
<Module: Attribute1, Module: Attribute2..., Module: AttributeN>
```

Dans le code source, les attributs globaux doivent être placés après les directives de niveau supérieur, telles que les instructions **Option Explicit** et **Imports**, mais avant toute déclaration d'espace de noms ou de type. Des attributs globaux peuvent apparaître dans plusieurs fichiers sources d'un projet, mais ils sont généralement placés dans le fichier AssemblyInfo.vb créé automatiquement avec les projets Visual Basic.

Les attributs d'assembly sont des valeurs qui fournissent des informations au sujet d'un assembly. Ils peuvent être répartis dans les catégories suivantes :

- attributs d'identité d'assembly ;
- attributs d'informations ;
- attributs de manifeste d'assembly ;
- attributs de nom fort ;

attributs d'identité d'assembly.

Trois attributs (associés à un nom fort s'il est applicable) déterminent l'identité d'un assembly : nom, version et culture. Ces attributs forment le nom complet de l'assembly et sont requis pour toute référence à celui-ci dans le code. Vous pouvez définir la version et la culture d'un assembly à l'aide d'attributs. La valeur de nom, cependant, est désignée par le compilateur, par l'IDE Visual Studio dans la Informations de l'assembly, boîte de dialogue ou par Assembly Linker (Al.exe) au moment de la création de l'assembly, d'après le fichier qui contient le manifeste d'assembly. L'attribut AssemblyFlagsAttribute détermine si plusieurs copies de l'assembly peuvent coexister.

Le tableau suivant répertorie les attributs d'identité.

Attribut	But
AssemblyName	Décrit intégralement l'identité d'un assembly.
AssemblyVersionAttribute	Spécifie la version d'un assembly.
AssemblyCultureAttribute	Spécifie la culture prise en charge par l'assembly.
AssemblyFlagsAttribute	Spécifie si un assembly prend en charge l'exécution côte à côte sur le même ordinateur, dans le mê me processus ou dans le même domaine d'application.

Le code suivant applique des attributs de version et de culture à un assembly :

VB

```
'Set version number for assembly.
<Assembly: Reflection.AssemblyVersionAttribute("4.3.2.1")>
```

```
'Set culture as German.
<Assembly: Reflection.AssemblyCultureAttribute("de")>
```

Attributs d'informations

Vous pouvez utiliser des attributs d'informations pour fournir des informations complémentaires relatives à la société ou au produit au sujet d'un assembly. Le tableau suivant répertorie les attributs d'informations définis dans l'espace de noms System.Reflection.

Attribut	But
AssemblyProductAttribute	Définit un attribut personnalisé qui spécifie un nom de produit pour un manifeste d'assembly.
AssemblyTrademarkAttribute	Définit un attribut personnalisé qui spécifie une marque pour un manifeste d'assembly.
AssemblyInformationalVersionAttribute	Définit un attribut personnalisé qui spécifie une version d'informations pour un manifeste d'assembly
AssemblyCompanyAttribute	Définit un attribut personnalisé qui spécifie un nom de société pour un manifeste d'assembly.
AssemblyCopyrightAttribute	Définit un attribut personnalisé qui spécifie un copyright pour un manifeste d'assembly.
AssemblyFileVersionAttribute	Indique au compilateur d'utiliser un numéro de version spécifique pour la ressource de version de fichier Win32.
CLSCompliantAttribute	Indique si l'assembly est conforme à la spécification CLS (Common Language Specification).

Attributs de manifeste d'assembly

Des attributs de manifeste d'assembly peuvent être employés pour fournir des informations dans le manifeste d'assembly, y compris son titre, une description, son alias par défaut et sa configuration. Le tableau suivant répertorie les attributs de manifeste d'assembly définis dans l'espace de noms **System.Reflection**.

Attribut	But
AssemblyTitleAttribute	Définit un attribut personnalisé qui spécifie un titre d'assembly pour un manifeste d'assembly.
AssemblyDescriptionAttribute	Définit un attribut personnalisé qui spécifie une description d'assembly pour un manifeste d'assembly.
AssemblyConfigurationAttribute	Définit un attribut personnalisé qui spécifie une configuration d'assembly (par exemple retail ou debug) pour un manifeste d'assembly.
AssemblyDefaultAliasAttribute	Définit un alias par défaut convivial pour un manifeste d'assembly.

Attributs de nom fort

Les noms forts sont des identificateurs uniques qui protègent l'identité et l'intégrité d'un assembly. Vous pouvez signer un assembly à partir de l'IDE Visual Studio à l'aide de la Page Signature, Concepteur de projets. Pour plus d'informations, consultez Gestion d'assembly et signature de manifeste.

Vous pouvez également utiliser des attributs avec nom fort afin de définir un nom fort pour un assembly. Le tableau suivant affiche les attributs avec nom fort définis dans l'espace de noms **System.Reflection**.

Attribut	But

AssemblyDelaySignAttribute	Valeur booléenne indiquant si vous devez réserver de l'espace dans un fichier exécutable pour la signature de nom fort, mais qui reporte la signature effective à un moment ultérieur. Pour plus d'informations, consultez Temporisation de signature d'un assembly.
AssemblyKeyFileAttribute	Indique le fichier contenant une clé. L'emplacement de KeyFile doit être relatif au répertoire de sortie du projet, c'est-à-dire %Répertoire du projet%\obj\<configuration>. Par exemple, si KeyFile se situe dans le répertoire de projet, vous devez spécifier l'attribut AssemblyKeyFile comme suit : [assembly: AssemblyKeyFile("..\\..\\mykey.snk")]
AssemblyKeyNameAttribute	Fait référence à une clé installée dans le fournisseur de services de chiffrement (CSP) sur votre ordinateur. Vous devez spécifier une clé pour signer votre fichier.

Si les valeurs KeyFile et KeyName sont spécifiées, le traitement suivant se produit :

- Si KeyName peut être trouvé dans le CSP, cette clé est utilisée.
- Si KeyName n'existe pas et si KeyFile existe, la clé de KeyFile est installée dans le CSP et est utilisée.
- Pour plus d'informations, consultez Aspects de la sécurité des assemblys.

Signatures des assemblys

Vous pouvez signer un assembly de deux manières différentes, mais complémentaires : avec un nom fort ou avec une signature. Vous pouvez le signer à l'aide de la Page Signature, Concepteur de projets de l'IDE Visual Studio ou en utilisant l'Outil File Signing Tool (Signcode.exe). La signature d'un assembly avec un nom fort ajoute un chiffrement à clé publique au fichier contenant le manifeste d'assembly. La signature avec un nom fort permet de vérifier l'unicité du nom, empêche l'usurpation de noms et fournit aux appelants une certaine identité lorsqu'une référence est résolue. Pour plus d'informations, consultez Gestion d'assembly et signature de manifeste et Comment : signer un assembly avec un nom fort.

Pour signer un assembly avec un nom fort

1. Utilisez le Outil Strong Name Tool (Sn.exe) pour générer un fichier de clé contenant une paire de clés.
2. Ajoutez des attributs de nom fort au fichier AssemblyInfo.vb créé automatiquement avec les projets Visual Basic. Pour modifier ce fichier, double-cliquez sur son nom dans l'**Explorateur de solutions**.

   L'exemple suivant utilise la temporisation de signature pour créer un assembly à nom fort avec un fichier de clé publique appelé `myKey.snk`.

   VB

```
<Assembly: Reflection.AssemblyKeyFile("myKey.snk")>
<Assembly: Reflection.AssemblyDelaySign(True)>
```

   Remarque

   Les attributs d'assembly sont ignorés si ce n'est pas un assembly que vous créez, comme c'est le cas lors de la compilation à partir de la ligne de commande avec l'option **/target:module**.

Voir aussi

Tâches

Comment : définir vos propres attributs
Comment : signer un assembly avec un nom fort

Référence

Outil Strong Name Tool (Sn.exe)
Outil File Signing Tool (Signcode.exe)
Page Signature, Concepteur de projets
Informations de l'assembly, boîte de dialogue

Concepts

Application des attributs

Aspects de la sécurité des assemblys

Autres ressources

Attributs personnalisés en Visual Basic

Gestion des propriétés de l'application

Gestion d'assembly et signature de manifeste

Attributs personnalisés en Visual Basic

Les attributs personnalisés sont des attributs définis par l'utilisateur qui fournissent des informations supplémentaires au sujet d'éléments de programme. Vous pouvez, par exemple, définir un attribut de sécurité personnalisé qui détermine les autorisations requises par l'appelant afin d'exécuter une procédure.

Les attributs personnalisés sont définis dans des classes d'attributs basées sur la classe System.Attribute. Les classes d'attributs, quant à elles, utilisent un attribut nommé AttributeUsageAttribute pour fournir des informations au sujet de la manière dont l'attribut peut être employé. Le fait de spécifier `Inherited = True` indique qu'un attribut peut se propager aux classes dérivées. Lorsque la propriété AllowMultiple a la valeur **True**, vous pouvez appliquer plusieurs instances de l'attribut à un élément de programme. Quant à l'énumération AttributeTargets, elle permet de définir les genres d'éléments de programme auxquels votre attribut peut s'appliquer.

Dans le code suivant, l'attribut **AttributeUsageAttribute** est un attribut qui peut être appliqué à n'importe quel type d'élément, qui peut être hérité, mais qui n'est applicable qu'une seule fois :

VB

```
<AttributeUsage(AttributeTargets.All, Inherited:=True, AllowMultiple:=False)> _
Class TestAttribute1
    Inherits Attribute
End Class
```

Vous pouvez utiliser l'opérateur **Or** pour combiner plusieurs éléments de l'énumération **AttributeTargets**, comme l'illustre le code suivant :

VB

```
<AttributeUsage(AttributeTargets.Class Or AttributeTargets.Method)> _
Class TestAttribute2
    Inherits Attribute
End Class
```

Dans cette section

Comment : définir vos propres attributs

Explique comment créer vos propres attributs à l'aide de classes d'attributs.

Comment : récupérer les attributs Custom

Explique comment récupérer des attributs personnalisés à l'aide de GetCustomAttribute ou de GetCustomAttributes.

Exemples d'utilisation d'attributs personnalisés

Fournit un exemple de code qui définit un attribut personnalisé pouvant être appliqué uniquement aux classes et montre comment utiliser ce nouvel attribut.

Rubriques connexes

Visual Basic et le .NET Framework

Décrit le rôle de Visual Basic dans le .NET Framework.

Programmation orientée objet dans Visual Basic

Fournit des informations au sujet de la programmation orientée objet et sur son utilisation.

Métadonnées et composants autodescriptifs

Fournit des informations détaillées sur les types de métadonnées utilisés dans Visual Studio, y compris sur les attributs.

Comment : définir vos propres attributs

Grâce aux classes d'attribut, vous pouvez créer vos propres attributs personnalisés et les utiliser conjointement avec les attributs .NET Framework pour fournir des informations supplémentaires sur des éléments de programme.

Pour définir un attribut personnalisé

1. Déclarez une classe et appliquez-lui l'attribut AttributeUsageAttribute. Le nom de votre classe est celui du nouvel attribut, comme illustré dans le code suivant :

   VB

   ```
   <AttributeUsage(AttributeTargets.All)> Class TestAttribute
   ```

2. Déclarez que la classe hérite de System.Attribute:

   VB

   ```
   Inherits System.Attribute
   ```

3. Définissez des champs **Private** qui stockeront des valeurs de propriété :

   VB

   ```
   Private m_SomeValue As String
   ```

4. Au besoin, créez un constructeur pour l'attribut :

   VB

   ```
   Public Sub New(ByVal Value As String)
       m_SomeValue = Value
   End Sub
   ```

5. Définissez des méthodes, des champs et des propriétés pour l'attribut :

   VB

   ```
   Public Sub Attr(ByVal AttrValue As String)
       'Add method code here.
   End Sub
   Public Property SomeValue() As String  ' A named parameter.
       Get
           Return m_SomeValue
       End Get
       Set(ByVal Value As String)
           m_SomeValue = Value
       End Set
   End Property
   ```

6. Terminez la classe par la construction **End Class** :

   VB

   ```
   End Class
   ```

Voir aussi

Référence

AttributeUsageAttribute

Concepts

Application des attributs

Durée de vie d'un objet : création et destruction des objets

Comment : récupérer les attributs Custom

Vous pouvez récupérer des attributs personnalisés à l'aide des méthodes GetCustomAttribute ou GetCustomAttributes de la classe Attribute.

Pour extraire une seule instance d'un attribut personnalisé d'une classe

1. Ajoutez une instruction **Imports** tout en haut du code source afin d'importer la classe **Attribute** à partir de l'espace de noms System :

   VB

   ```
   Imports System.Attribute
   ```

2. Créez une procédure destinée à extraire l'attribut :

   VB

   ```
   Sub RetrieveAttribute()

   End Sub
   ```

3. Dans cette procédure, déclarez une variable de type **Attribute** et une autre de même type que l'attribut à extraire :

   VB

   ```
   Dim Attr As Attribute
   Dim CustAttr As CustomAttribute
   ```

4. Utilisez l'opérateur **GetType** pour passer le type de la classe et l'attribut à un appel à la méthode **GetCustomAttribute**, puis assignez la valeur retournée à la variable déclarée comme **Attribute** :

   VB

   ```
   Attr = GetCustomAttribute(Me.GetType, _
                             GetType(CustomAttribute), False)
   ```

5. Utilisez la fonction **CType** pour convertir le type de données de l'attribut de l'attribut générique à l'attribut spécifique du type extrait. Assignez ensuite le résultat à la variable déclarée comme le type d'attribut personnalisé :

   VB

   ```
   CustAttr = CType(Attr, CustomAttribute)
   ```

6. Vérifiez si l'attribut a été extrait. Si c'est le cas, utilisez ses champs, ses propriétés et ses méthodes :

   VB

   ```
   If CustAttr Is Nothing Then
       MsgBox("The attribute was not found.")
   Else
       'Get the label and value from the custom attribute.
       MsgBox("The attribute label is: " & CustAttr.Label)
       MsgBox("The attribute value is: " & CustAttr.Value)
   End If
   ```

Dans l'exemple ci-dessus, la procédure `RetrieveAttribute` appelle la méthode **GetCustomAttribute** de la classe **System.Attribute** pour obtenir l'attribut personnalisé appliqué à la classe `ThisClass`. **GetCustomAttribute** étant une méthode partagée, il n'est pas nécessaire de créer d'abord une instance de **System.Attribute**. La fonction **CType** convertit l'attribut retourné du type **System.Attribute** au type d'attribut personnalisé `CustomAttribute`.

Voir aussi

Tâches

Comment : définir vos propres attributs

Référence

GetType, opérateur
CType, fonction
IsNothing, fonction
GetAttr, fonction
GetCustomAttribute
GetCustomAttributes
AttributeUsageAttribute

Concepts

Application des attributs
Récupération des informations stockées dans les attributs

Exemples d'utilisation d'attributs personnalisés

L'exemple suivant définit un attribut personnalisé qui peut être appliqué uniquement aux classes :

Exemple

VB

```
<AttributeUsage(AttributeTargets.Class)> Public Class CustomAttribute
    Inherits System.Attribute

    'Declare two private fields to store the property values.
    Private m_LlabelValue As String
    Private m_VValueValue As Integer

    'The Sub New constructor is the only way to set the properties.
    Public Sub New(ByVal _Label As String, ByVal _Value As Integer)
        m_LlabelValue = _Label
        m_VValueValue = _Value
    End Sub

    Public ReadOnly Property Label() As String
        Get
            Return m_LlabelValue
        End Get
    End Property

    Public ReadOnly Property Value() As Integer
        Get
            Return m_VValueValue
        End Get
    End Property
End Class
```

Seul le constructeur de la classe d'attributs peut définir les propriétés que contient cet attribut. Le code ci-dessous indique comment utiliser l'attribut :

VB

```
' Apply the custom attribute to this class.
<Custom("Some metadata", 66)> Class ThisClass
    ' Add class members here.
End Class
```

Voir aussi

Tâches

Comment : définir vos propres attributs

Comment : récupérer les attributs Custom

Référence

AttributeUsageAttribute

Concepts

Application des attributs

Récupération des informations stockées dans les attributs

Comment : appliquer plusieurs attributs

Cet exemple montre comment appliquer plusieurs attributs au même élément.

Vous pouvez spécifier plusieurs attributs dans le même bloc d'attributs en séparant chaque attribut par une virgule.

Exemple

Cet exemple montre comment appliquer l'attribut EditorBrowsableAttribute et l'attribut ObsoleteAttribute à la même fonction.

VB

```
<System.ComponentModel.EditorBrowsable( _
    System.ComponentModel.EditorBrowsableState.Never), _
Obsolete("This method should not be used.")> _
Public Shadows Function Update(ByVal x As Integer) As Integer
    ' The function code goes here.
End Function
```

Voir aussi

Concepts

Vue d'ensemble des attributs dans Visual Basic

Application des attributs

Autres ressources

Attributs en Visual Basic

Éléments déclarés en Visual Basic

Un *élément déclaré* est un élément de programmation qui est défini dans une instruction de déclaration. Les éléments déclarés comprennent notamment des variables, des constantes, des énumérations, des classes, des structures, des modules, des interfaces, des procédures, des paramètres de procédure, des retours de fonction, des références de procédure externes, des opérateurs, des propriétés, des événements et des délégués.

Les instructions de déclaration contiennent les éléments suivants :

- Dim, instruction (Visual Basic)
- Const, instruction (Visual Basic)
- Enum, instruction (Visual Basic)
- Class, instruction (Visual Basic)
- Structure, instruction
- Module, instruction
- Interface, instruction (Visual Basic)
- Function, instruction (Visual Basic)
- Sub, instruction (Visual Basic)
- Declare, instruction
- Operator, instruction
- Property, instruction
- Event, instruction
- Delegate, instruction

Dans cette section

Noms d'éléments déclarés

Explique comment nommer les éléments et utiliser la casse alphabétique.

Caractéristiques d'éléments déclarés

Présente les caractéristiques telles que la portée, que possèdent les éléments déclarés.

Références aux éléments déclarés

Explique comment un compilateur établit une correspondance entre une référence et une déclaration et comment qualifier un nom.

Rubriques connexes

Programmation orientée objet dans Visual Basic

Traite de la programmation orientée objet.

Structure de programme et conventions de codage

Présente les indications pour faciliter la lecture, la compréhension et la maintenance du code.

Instructions de déclaration en Visual Basic

Décrit les instructions permettant de nommer et de définir des procédures, des variables, des tableaux et des constantes.

Contextes de déclaration et niveaux d'accès par défaut

Répertorie les types d'éléments déclarés et affiche pour chacun d'eux son instruction de déclaration, son contexte de déclaration et son niveau d'accès par défaut.

Noms d'éléments déclarés

Chaque élément déclaré a un nom, également appelé *identificateur*, utilisé par le code pour faire référence à l'élément déclaré.

Règles

Un nom d'élément dans Visual Basic doit observer les règles suivantes :

- commencer par un caractère alphabétique ou un trait de soulignement (_) ;
- contenir uniquement des caractères alphabétiques, des chiffres décimaux et des traits de soulignement ;
- comprendre au moins un caractère alphabétique ou un chiffre décimal s'il commence par un trait de soulignement ;
- être inférieur ou égal à 1 023 caractères.

La longueur maximale de 1 023 caractères s'applique également à un nom qualifié complet, tel que : `outerNamespace.middleNamespace.innerNamespace.thisClass.thisElement`.

Vous trouverez ci-dessous des exemples de noms d'éléments valides.

```
aB123__45

_567
```

Vous trouverez ci-dessous des exemples de noms d'éléments non valides. Le premier contient uniquement un trait de soulignement, le second commence par un chiffre décimal et le troisième contient un caractère non valide ($).

```
' Three INVALID element names

_

12ABC

xyz$wv
```

Attention

Les noms d'élément commençant par un trait de soulignement (_) ne font pas partie de la spécification CLS (Spécification CLS (Common Language Specification)) ; par conséquent, le code conforme CLS ne peut pas utiliser de compo sant qui définit ce type de noms. Toutefois, un trait de soulignement inséré à toute autre position dans un nom d'élément est conforme à la spécification CLS.

Indications sur la longueur du nom

D'un point de vue pratique, votre nom doit être aussi court que possible tout en identifiant clairement la nature de l'élément. La lisibilité de votre code s'en trouve améliorée, et la justification et la taille de fichier source réduites.

En revanche, votre nom ne doit pas être court au point de ne pas décrire suffisamment ce que l'élément représente et comment votre code l'utilise. Cette considération est importante pour la lisibilité de votre code. Si une autre personne essaie de le comprendre ou si vous l'examinez longtemps après l'avoir écrit, des noms d'éléments appropriés peuvent vous faire gagner un temps précieux.

Noms d'échappement

En règle générale, un nom d'élément ne doit correspondre à aucun des mots clés réservés par Visual Basic, tels que **Case** ou **Friend**. Toutefois, vous pouvez définir un *nom d'échappement* placé entre crochets (**[ ]**). Un nom d'échappement peut correspondre à n'importe quel mot clé Visual Basic puisque la présence des crochets lève toute ambiguïté. Vous pouvez également utiliser les crochets lorsque vous faites référence au nom ultérieurement dans votre code.

En général, vous devez utiliser des noms échappés uniquement dans les cas suivants :

- Votre code a effectué une migration d'une version antérieure de Visual Basic qui n'a pas réservé le mot clé utilisé comme un nom ; ou
- vous utilisez du code écrit dans un autre langage dans lequel le mot clé donné n'est pas réservé.

Sinon, vous devez envisager de renommer l'élément si son nom est en conflit avec un mot clé. L'environnement de développement intégré (IDE) offre un moyen simple d'effectuer cette opération. Pour plus d'informations, consultez Comment : renommer un identificateur.

Respect de la casse dans les noms

Les noms d'éléments dans Visual Basic ne respectent pas la casse. En d'autres termes, lorsque le compilateur compare deux noms qui ne se distinguent que par la casse, il les interprète comme un seul et même nom. Il considère, par exemple, que `ABC` et `abc` font référence au même élément déclaré.

Néanmoins, le Common Language Runtime (CLR) utilise des liaisons qui respectent la casse. En conséquence, lorsque vous générez un assembly ou une DLL et que vous les mettez à la disposition d'autres assemblys, vos noms respectent alors la casse. Par exemple, si vous définissez une classe avec un élément appelé `ABC` et que d'autres assemblys utilisent votre classe via le Common Language Runtime, ils doivent faire référence à l'élément sous la forme `ABC`. Si, par la suite, vous recompilez votre classe et que vous changez le nom de l'élément en `abc`, les autres assemblys qui utilisent votre classe ne peuvent plus accéder à cet élément. Dès lors, lorsque vous libérez une version mise à jour d'un assembly, évitez de modifier la casse des éléments publics.

Noms et paramètres régionaux

La comparaison de noms est indépendante des paramètres régionaux. Si deux noms correspondent dans des paramètres régionaux spécifiques, ils correspondent dans tous.

Voir aussi

Concepts

Caractéristiques d'éléments déclarés
Instructions de déclaration en Visual Basic

Autres ressources

Éléments déclarés en Visual Basic
Références aux éléments déclarés

Caractéristiques d'éléments déclarés

Une *caractéristique* d'un élément déclaré est un aspect de cet élément qui détermine l'interaction du code avec ce dernier. Chaque élément déclaré possède une ou plusieurs des caractéristiques suivantes qui lui sont associées :

- *Type de données* — les valeurs possibles de l'élément et le mode de stockage de ces valeurs. Pour plus d'informations, consultez Liste des types de données (Visual Basic).
- *Durée de vie* — la période d'exécution pendant laquelle il est possible d'utiliser l'élément. Pour plus d'informations, consultez Comment : augmenter la durée de vie d'une variable.
- *Portée* — l'ensemble du code qui peut faire référence à l'élément sans qualifier son nom. Pour plus d'informations, consultez Comment : contrôler la portée d'une variable.
- *Niveau d'accès* — l'autorisation pour le code d'utiliser l'élément. Pour plus d'informations, consultez Comment : contrôler la disponibilité d'une variable.

Caractéristiques des éléments

Le tableau suivant répertorie les éléments déclarés et les caractéristiques qui s'appliquent à chacun d'eux.

Élément	Type de données	Durée de vie	Portée [1]	Niveau d'accès
Variable	Oui	Oui	Oui	Oui
Constante	Oui	Non	Oui	Oui
Énumération	Oui	Non	Oui	Oui
Structure	Non	Non	Oui	Oui
Propriété	Oui	Oui	Oui	Oui
Méthode	Non	Oui	Oui	Oui
Procédure (**Sub** ou **Function**)	Non	Oui	Oui	Oui
Paramètre de procédure	Oui	Oui	Oui	Non
Retour de fonction	Oui	Oui	Oui	Non
Opérateur	Oui	Non	Oui	Oui
Interface	Non	Non	Oui	Oui
Classe	Non	Non	Oui	Oui
Événement	Non	Non	Oui	Oui
Délégué	Non	Non	Oui	Oui

La portée [1] est parfois connue sous le nom de *visibilité*.

Voir aussi

Concepts

Noms d'éléments déclarés
Durée de vie dans Visual Basic
Portée dans Visual Basic

Niveaux d'accès dans Visual Basic
Types de données en Visual Basic
Déclaration de variable dans Visual Basic

Autres ressources

Éléments déclarés en Visual Basic
Références aux éléments déclarés

Durée de vie dans Visual Basic

La *durée de vie* d'un élément déclaré est la période pendant laquelle il est accessible et peut être utilisé. Les variables sont les seuls éléments qui ont une durée de vie. Pour cette raison, le compilateur traite les paramètres de procédure et les retours de fonction comme des cas spéciaux de variables. La durée de vie d'une variable représente la période pendant laquelle elle peut stocker une valeur. La variable peut voir sa valeur varier au cours de sa durée de vie, mais elle stocke toujours une valeur quelconque.

Durées de vie différentes

Une *variable membre* (déclarée au niveau du module, à l'extérieur de toute procédure) a en général le même durée de vie que l'élément dans laquelle elle est déclarée. Une variable non partagée déclarée dans une classe ou une structure existe en tant que copie séparée pour chaque instance de la classe ou de la structure dans laquelle elle est déclarée. Chaque variable de ce genre a la même durée de vie que son instance. Toutefois, une variable **Shared** a une seule durée de vie qui correspond à la durée d'exécution de votre application.

Une *variable locale* (déclarée à l'intérieur d'une procédure) existe uniquement tant que la procédure dans laquelle elle est déclarée s'exécute. C'est également le cas des paramètres de cette procédure et de tout retour de fonction. Toutefois, si la procédure appelle d'autres procédures, les variables locales conservent leurs valeurs pendant la durée d'exécution des procédures appelées.

Début de la durée de vie

La durée de vie d'une variable locale commence au moment où le contrôle pénètre dans la procédure dans laquelle elle est déclarée. Chaque variable locale est initialisée avec la valeur par défaut de son type de données dès que la procédure est exécutée. Lorsque la procédure rencontre une Dim, instruction (Visual Basic) qui spécifie des valeurs initiales, elle définit ces variables à ces valeurs, même si votre code leur avait déjà assigné d'autres valeurs.

Chaque membre d'une variable structure est initialisé comme s'il s'agissait d'une variable distincte. De la même façon, chaque élément d'une variable tableau est initialisé individuellement.

Les variables déclarées dans un bloc à l'intérieur d'une procédure (tel qu'une boucle **For**) sont initialisées lors de l'entrée dans la procédure. Ces initialisations entrent en vigueur même si votre code n'exécute pas le bloc.

Fin de la durée de vie

Lorsqu'une procédure se termine, les valeurs de ses variables locales ne sont pas conservées et Visual Basic libère leur mémoire. La prochaine fois que vous appelez la procédure, toutes ses variables locales sont recréées et réinitialisées.

Lorsqu'une instance d'une classe ou d'une structure prend fin, ses variables non partagées perdent leur mémoire et leur valeur. Chaque nouvelle instance de la classe ou de la structure crée et réinitialise ses variables non partagées. Toutefois, les variables **Shared** sont conservées jusqu'au terme de l'exécution de votre application.

Extension de la durée de vie

Si vous déclarez une variable locale avec le mot clé Static (Visual Basic), sa durée de vie est plus longue que le temps d'exécution de sa procédure. Le tableau suivant montre comment la déclaration de procédure détermine la durée de vie d'une variable **Static**.

Emplacement de procédure et partage	**La durée de vie de variable statique commence**	**La durée de vie de variable statique se termine**
Dans un module (partagé par défaut)	Au premier appel de la procédure	À l'arrêt de l'exécution de votre application
Dans une classe ou une structure, **Shared** (la procédure n'est pas un membre d'instance)	Au premier appel de la procédure sur une instance spécifique ou sur le nom de classe ou de structure lui-même	À l'arrêt de l'exécution de votre application
Dans une instance d'une classe ou d'une structure, non **Shared** (la procédure est un membre d'instance)	Au premier appel de la procédure sur une instance spécifique	Lorsque l'instance est libérée pour le garbage collection (GC)

Voir aussi

Tâches

Comment : augmenter la durée de vie d'une variable

Dépannage des types de données

Référence

Shared (Visual Basic)

Nothing (Visual Basic)

Concepts

Noms d'éléments déclarés

Portée dans Visual Basic

Niveaux d'accès dans Visual Basic

Variables en Visual Basic

Déclaration de variable dans Visual Basic

Autres ressources

Références aux éléments déclarés

Comment : augmenter la durée de vie d'une variable

Normalement, une variable cesse d'exister lorsque l'élément de programmation dans lequel elle est déclarée cesse d'exister. Toutefois, vous pouvez augmenter sa durée de vie pour qu'elle soit plus longue que l'élément la contenant en la déclarant à l'aide du mot clé Static (Visual Basic).

Pour plus d'informations, consultez Durée de vie dans Visual Basic.

Pour faire vivre une variable plus longtemps que l'élément la contenant

- Si la variable est une *variable locale* (déclarée à l'intérieur d'une procédure), insérez le mot clé **Static** dans Dim, instruction (Visual Basic) qui la déclare. Une variable statique existe aussi longtemps que la classe ou le module contenant la procédure qui déclare la variable.

  Si une variable locale se trouve à l'intérieur d'une procédure Shared (Visual Basic), la procédure et ses variables existent aussi longtemps que votre application s'exécute. N'utilisez pas **Static** dans ce cas.

- Si la variable est une *variable membre* (déclarée dans une classe ou une structure à l'extérieur de toute procédure), insérez le mot clé **Shared** dans l'instruction **Dim** qui la déclare. Une variable commune n'est associée à aucune instance de sa classe ni de sa structure, et elle existe aussi longtemps que votre application s'exécute. Vous ne pouvez pas utiliser **Static** sur une variable membre.

  Si une variable membre est déclarée dans un module, elle existe toujours aussi longtemps que votre application s'exécute. N'utilisez pas **Shared** dans ce cas.

Exemple

L'exemple suivant déclare une variable à l'aide du mot clé Static (Visual Basic). (Notez que vous n'avez pas besoin du mot clé **Dim** lorsque Dim, instruction (Visual Basic) utilise un modificateur, tel que **Static**.)

```
Function runningTotal(ByVal num As Integer) As Integer
    Static applesSold As Integer
    applesSold = applesSold + num
    Return applesSold
End Function
```

Dans l'exemple précédent, la variable `applesSold` continue d'exister après que la procédure `runningTotal` retourne au code appelant. La prochaine fois que `runningTotal` est appelé, `applesSold` conserve sa valeur précédemment calculée.

Si `applesSold` avait été déclaré sans utiliser **Static**, les valeurs précédemment accumulées n'auraient pas été conservées d'un appel à l'autre à `runningTotal`. Au prochain appel de `runningTotal`, `applesSold` aurait été recréé et initialisé à 0 et `runningTotal` aurait simplement retourné la même valeur que celle avec laquelle elle avait été appelée.

Compilation du code

Vous pouvez initialiser la valeur d'une variable locale statique dans sa déclaration. Si vous déclarez un tableau **Static**, vous pouvez initialiser son rang (nombre de dimensions), la longueur de chaque dimension et les valeurs de chaque élément.

Variables statiques du même nom

Vous pouvez déclarer des variables statiques avec le même nom dans plusieurs procédures. Dans ce cas, le compilateur Visual Basic considère chacune des variables comme un élément distinct. L'initialisation d'une de ces variables n'affecte pas les valeurs des autres. C'est également vrai si vous définissez une procédure avec un ensemble de surcharges et que vous déclarez une variable statique avec le même nom dans chaque surcharge.

Éléments conteneurs pour les variables statiques

Vous pouvez déclarer une variable locale statique dans une classe, c'est-à-dire dans une procédure au sein de cette classe. Toutefois, vous ne pouvez pas déclarer une variable locale statique dans une structure, ni comme membre de structure ni comme variable locale d'une procédure dans cette structure.

Sécurité

Dans l'exemple précédent, vous pouvez obtenir la même durée de vie en déclarant `applesSold` au niveau du module. Toutefois, si vous avez modifié de la sorte la portée d'une variable, la procédure ne dispose plus d'un accès exclusif à celle-ci. Dans la mesure où d'autres procédures peuvent accéder à `applesSold` et modifier sa valeur, le total évolutif n'est plus fiable et le code

est plus difficile à gérer.

Voir aussi

Concepts

Durée de vie dans Visual Basic
Caractéristiques d'éléments déclarés
Portée dans Visual Basic
Niveaux d'accès dans Visual Basic
Variables en Visual Basic
Déclaration de variable dans Visual Basic
Surcharge de procédure

Portée dans Visual Basic

La *portée* d'un élément déclaré est l'ensemble du code qui peut faire référence à cet élément sans qualifier son nom ou sans le rendre disponible via une instruction Imports. Il existe différents niveaux de portée d'un élément :

Niveau	Description
Portée de bloc	Élément disponible uniquement dans le bloc de code dans lequel il est déclaré.
Portée de procédure	Disponible pour tout le code dans la procédure dans laquelle il est déclaré
Portée de module	Élément disponible pour tout le code au sein du module, de la classe ou de la structure dans laquelle il est déclaré.
Portée d'espace de noms	Disponible pour tout le code dans l'espace de noms dans lequel il est déclaré

Ces niveaux de portée vont du plus restreint (bloc) au plus large (espace de noms). La *plus petite portée* est le plus petit ensemble de code qui peut référencer l'élément sans le qualifier. Pour plus d'informations, consultez « Niveaux de portée » dans cette page.

Spécifier la portée et définir des variables

Vous spécifiez la portée d'un élément lorsque vous le déclarez. Les facteurs suivants sont susceptibles d'affecter la portée :

- La région (bloc, procédure, module, classe ou structure) dans laquelle vous déclarez l'élément ;
- L'espace de noms contenant la déclaration de l'élément ;
- Niveau d'accès que vous déclarez pour l'élément

Soyez très prudent lorsque vous définissez des variables avec le même nom, mais une portée différente ; vous pouvez être confronté à des résultats inattendus. Pour plus d'informations, consultez
Résolution d'une référence lorsque plusieurs variables ont le même nom.

Niveaux de portée

Un élément de programmation est disponible dans toute la région dans laquelle vous le déclarez. Tout le code d'une même région peut faire référence à l'élément sans qualifier son nom.

Portée de bloc

Un bloc est un jeu d'instructions délimité par des instructions de déclaration de début et de fin, telles que les éléments suivants :

- **Do** et **Loop**
- **For** [**Each**] et **Next**
- **If** et **End If**
- **Select** et **End Select**
- **SyncLock** et **End SyncLock**
- **Try** et **End Try**
- **While** et **End While**
- **With** et **End With**

Si vous déclarez une variable dans un bloc, vous pouvez l'utiliser uniquement dans ce bloc. Dans l'exemple suivant, la portée de la variable de type entier `cube` est le bloc compris entre **If** et **End If**, et vous ne pouvez plus faire référence à `cube` lorsque l'exécution passe en dehors du bloc.

```
If n < 1291 Then
    Dim cube As Integer
    cube = n ^ 3
End If
```

Remarque

Même si la portée d'une variable est limitée à un bloc, sa durée de vie correspond néanmoins à celle de toute la procédure. Si vous entrez dans le bloc plusieurs fois au cours de la procédure, chaque variable de bloc conserve sa valeur précédente. Dans un tel cas, vous pouvez éviter la présence de résultats inattendus en initialisant les variables de bloc au début de chaque bloc.

Portée de procédure

Un élément déclaré dans une procédure n'est pas disponible en dehors de celle-ci. Seule la procédure qui contient la déclaration peut l'utiliser. Les variables à ce niveau sont également appelées des *variables locales*. Vous les déclarez à l'aide de Dim, instruction (Visual Basic), avec ou sans le mot clé Static (Visual Basic).

Les portées de bloc et de procédure sont étroitement liées. Si vous déclarez une variable à l'intérieur d'une procédure, mais en dehors de tout bloc de cette procédure, la variable peut être considérée comme ayant une portée de bloc, le bloc représentant toute la procédure.

Remarque

Tous les éléments locaux, même s'il s'agit de variables **Static**, sont privés dans la procédure dans laquelle ils apparaissent. V ous ne pouvez pas déclarer d'élément à l'aide du mot clé Public (Visual Basic) dans une procédure.

Portée de module

Pour des raisons pratiques, le terme *niveau de module* est appliqué indifféremment aux modules, aux classes et aux structures. Vous pouvez déclarer des éléments à ce niveau en plaçant l'instruction de déclaration à l'extérieur de toute procédure ou tout bloc figurant dans un module, une classe ou une structure.

Lorsque vous faites une déclaration au niveau du module, le niveau d'accès que vous choisissez détermine la portée. L'espace de noms qui contient le module, la classe ou la structure affecte également la portée.

Les éléments que vous déclarez avec un niveau d'accès Private (Visual Basic) sont disponibles pour chaque procédure de ce module, mais pas pour le code d'un autre module. L'instruction **Dim** au niveau du module a la valeur **Private** par défaut si vous n'utilisez aucun mot clé de niveau d'accès. Toutefois, vous pouvez faire en sorte que la portée et le niveau d'accès soient plus évidents en utilisant le mot clé **Private** dans l'instruction **Dim**.

Dans l'exemple précédent, toutes les procédures définies dans le module peuvent faire référence à la variable chaîne `strMsg`. Lors de l'appel de la seconde procédure, celle-ci affiche le contenu de la variable chaîne `strMsg` dans une boîte de dialogue.

```
' Put the following declaration at module level (not in any procedure).
Private strMsg As String
' Put the following Sub procedure in the same module.
Sub initializePrivateVariable()
    strMsg = "This variable cannot be used outside this module."
End Sub
' Put the following Sub procedure in the same module.
Sub usePrivateVariable()
    MsgBox(strMsg)
End Sub
```

Portée d'espace de noms

Si vous déclarez un élément au niveau du module à l'aide du mot clé Friend (Visual Basic) ou Public (Visual Basic), il est accessible à toutes les procédures dans tout l'espace de noms dans lequel l'élément est déclaré. Avec la modification suivante apportée à l'exemple ci-dessus, la variable chaîne `strMsg` peut faire l'objet d'une référence par le code n'importe où dans l'espace de noms de sa déclaration.

```
' Include this declaration at module level (not inside any procedure).
Public strMsg As String
```

La portée d'espace de noms inclut des espaces de noms imbriqués. Un élément disponible dans un espace de noms l'est également dans tout espace de noms imbriqué dans le précédent.

Si votre projet ne contient pas d'instructions Namespace, tout le contenu du projet appartient au même espace de noms. Dans ce cas, la portée d'espace de noms peut être considérée comme portée de projet. Les éléments **Public** dans un module, une classe ou une structure sont également disponibles pour tout projet qui référence leur projet.

Choix de portée

Lorsque vous déclarez une variable, tenez compte des points suivants lors du choix de sa portée.

Avantages offerts par les variables locales

Les variables locales sont un bon choix pour tous les types de calculs temporaires, pour les raisons suivantes :

- **Capacité à éviter les conflits de noms.** Les noms de variables locales ne risquent pas d'entrer en conflit. Par exemple, vous pouvez créer plusieurs procédures différentes qui contiennent une variable appelée `intTemp`. Tant que chaque variable `intTemp` est déclarée comme variable locale, chaque procédure ne reconnaît que sa propre version de la variable `intTemp`. N'importe quelle procédure peut modifier la valeur de sa variable locale `intTemp` sans affecter les variables `intTemp` contenues dans les autres procédures.

- **Consommation de mémoire.** Les variables locales consomment de la mémoire que lorsque leur procédure s'exécute. Leur mémoire est diffusée lorsque la procédure retourne au code appelant. Par contraste, les variables Shared (Visual Basic) et Static (Visual Basic) consomment des ressources mémoire tant que votre application est en cours d'exécution, aussi utilisez-les uniquement lorsque cela est nécessaire. Les *variables d'instance* consomment de la mémoire au cours de la durée de vie de leur instance, ce qui les rend moins efficaces que des variables locales, mais potentiellement plus efficaces que les variables **Shared** ou **Static**.

Réduire la portée

En règle générale, lorsque vous déclarez une variable ou une constante, il est conseillé de définir une portée aussi restreinte que possible (la portée de bloc est la plus petite). Cela permet de conserver de la mémoire et de réduire les risques que votre code fasse référence à la mauvaise variable. De même, ne déclarez une variable de type Static (Visual Basic) que lorsque sa valeur doit être conservée entre les appels de procédure.

Voir aussi

Tâches

Comment : contrôler la portée d'une variable

Concepts

Caractéristiques d'éléments déclarés
Durée de vie dans Visual Basic
Niveaux d'accès dans Visual Basic
Résolution d'une référence lorsque plusieurs variables ont le même nom
Déclaration de variable dans Visual Basic

Comment : contrôler la portée d'une variable

Normalement, une variable figure dans la *portée* ou est visible par référence dans l'ensemble de la région dans laquelle vous la déclarez. Dans certains cas, le *niveau d'accès* de la variable peut influencer sa portée.

Pour plus d'informations, consultez Portée dans Visual Basic.

Portée au niveau du bloc ou de la procédure

Pour rendre une variable visible uniquement dans un bloc

- Placez le Dim, instruction (Visual Basic) de la variable entre les instructions de déclaration de début et de fin de ce bloc, par exemple entre les instructions **For** et **Next** d'une boucle **For**.

  Vous pouvez faire référence à la variable uniquement à partir du bloc.

Pour rendre une variable visible uniquement dans une procédure

- Placez l'instruction **Dim** de la variable à l'intérieur de la procédure, mais à l'extérieur de tout bloc (tel qu'un bloc **With**...**End With**).

  Vous pouvez faire référence à la variable uniquement à partir de la procédure, y compris à l'intérieur de tout bloc contenu dans la procédure.

Portée au niveau du module ou de l'espace de noms

Pour des raisons pratiques, le terme *niveau de module* est appliqué indifféremment aux modules, aux classes et aux structures. Le niveau d'accès d'une variable du niveau du module détermine sa portée. L'espace de noms qui contient le module, la classe ou la structure influence également la portée.

Pour rendre une variable visible dans l'ensemble d'un module, d'une classe ou d'une structure

1. Placez l'instruction **Dim** de la variable à l'intérieur du module, de la classe ou de la structure, mais à l'extérieur de toute procédure.
2. Incluez le mot clé Private (Visual Basic) dans l'instruction **Dim**.
3. Vous pouvez faire référence à la variable de n'importe quel endroit du module, de la classe ou de la structure, mais pas à l'extérieur de ceux-ci.

Pour rendre une variable visible dans l'ensemble d'un espace de noms

1. Placez l'instruction **Dim** de la variable à l'intérieur du module, de la classe ou de la structure, mais à l'extérieur de toute procédure.
2. Insérez le mot clé Friend (Visual Basic) ou Public (Visual Basic) dans l'instruction **Dim**.
3. Vous pouvez faire référence à la variable de n'importe quel endroit de l'espace de noms qui contient le module, la classe ou la structure.

Exemple

L'exemple suivant déclare une variable au niveau du module et limite sa visibilité au code dans le module.

```
Module demonstrateScope
    Private strMsg As String
    Sub initializePrivateVariable()
        strMsg = "This variable cannot be used outside this module."
    End Sub
    Sub usePrivateVariable()
        MsgBox(strMsg)
    End Sub
End Module
```

Dans l'exemple précédent, toutes les procédures définies dans le module `demonstrateScope` peuvent faire référence à la variable **String** `strMsg`. Lors de l'appel de la procédure `usePrivateVariable`, celle-ci affiche le contenu de la variable chaîne `strMsg` dans une boîte de dialogue.

Avec la modification suivante apportée à l'exemple ci-dessus, la variable chaîne `strMsg` peut faire l'objet d'une référence par le code n'importe où dans l'espace de noms de sa déclaration :

```
    Public strMsg As String
```

Programmation fiable

Plus la portée d'une variable est limitée, moins vous risquez d'y faire référence par erreur à la place d'une autre variable du même nom. Vous pouvez également réduire les problèmes de correspondance de référence.

Sécurité

Plus la portée d'une variable est limitée, moins vous risquez son utilisation incorrecte par du code malveillant .

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Portée dans Visual Basic
Durée de vie dans Visual Basic
Niveaux d'accès dans Visual Basic
Variables en Visual Basic
Déclaration de variable dans Visual Basic

Promotion de type

Lorsque vous déclarez un élément de programmation dans un module, Visual Basic élève sa portée vers l'espace de noms contenant le module. Ce concept porte le nom de *promotion de type*.

L'exemple suivant montre une définition squelette d'un module et de deux membres de ce module.

```
Namespace projNamespace
    Module projModule
        Public Enum basicEnum As Integer
            one = 1
            two = 2
        End Enum
        Public Class innerClass
            Public Sub numberSub(ByVal firstArg As Integer)
            End Sub
        End Class
    End Module
End Namespace
```

Dans `projModule`, les éléments de programmation déclarés au niveau du module sont promus vers `projNamespace`. Dans l'exemple précédent, `basicEnum` et `innerClass` sont promus, mais `numberSub` ne l'est pas, parce qu'il n'est pas déclaré au niveau du module.

Effet de la promotion de type

Grâce à la promotion de type, une chaîne de qualification n'a pas besoin d'inclure le nom du module. L'exemple suivant effectue deux appels à la procédure de l'exemple précédent.

```
Sub usePromotion()
    projNamespace.projModule.innerClass.numberSub(projNamespace.projModule.basicEnum.one)
    projNamespace.innerClass.numberSub(projNamespace.basicEnum.two)
End Sub
```

Dans l'exemple précédent, le premier appel utilise des chaînes de qualification complètes. Toutefois, cela n'est pas nécessaire du fait de la promotion de type. Le deuxième appel accède également aux membres du module sans inclure `projModule` dans les chaînes de qualification.

Invalidation de la promotion de type

Si l'espace de noms possède déjà un membre portant le même nom qu'un membre de module, la promotion de type est invalidée pour ce membre de module. L'exemple suivant montre une définition squelette d'une énumération et d'un module dans le même espace de noms.

```
Namespace thisNamespace
    Public Enum abc
        first = 1
        second
    End Enum
    Module thisModule
        Public Class abc
            Public Sub abcSub()
            End Sub
        End Class
        Public Class xyz
            Public Sub xyzSub()
            End Sub
        End Class
    End Module
End Namespace
```

Dans l'exemple précédent, Visual Basic ne peut pas promouvoir la classe `abc` vers `thisNameSpace` parce qu'il existe déjà une

énumération du même nom au niveau de l'espace de noms. Pour accéder à `abcSub`, vous devez utiliser la chaîne de qualification complète `thisNamespace.thisModule.abc.abcSub`. Toutefois, la classe `xyz` est promue et vous pouvez accéder à `xyzSub` à l'aide de la chaîne de qualification plus courte `thisNamespace.xyz.xyzSub`.

Invalidation de la promotion de type pour les types partiels

Si une classe ou une structure à l'intérieur d'un module utilise le mot clé Partial (Visual Basic), la promotion de type est automatiquement invalidée pour cette classe ou cette structure, que l'espace de noms possède ou non un membre du même nom. D'autres éléments dans le module restent disponibles pour la promotion de type.

Conséquences. L'invalidation de la promotion de type d'une définition partielle peut provoquer des résultats inattendus et même des erreurs de compilation. L'exemple suivant montre des définitions squelettes partielles d'une classe, dont l'une se trouve à l'intérieur d'un module.

```
Namespace sampleNamespace
    Public Partial Class sampleClass
        Public Sub sub1()
        End Sub
    End Class
    Module sampleModule
        Public Partial Class sampleClass
            Public Sub sub2()
            End Sub
        End Class
    End Module
End Namespace
```

Dans l'exemple précédent, le développeur peut s'attendre à ce que le compilateur fusionne les deux définitions partielles de `sampleClass`. Toutefois, le compilateur n'envisage pas de promotion pour la définition partielle à l'intérieur de `sampleModule`. En conséquence, il essaie de compiler deux classes distinctes et séparées, intitulées `sampleClass`, mais avec des chemins d'accès de qualification différents.

Le compilateur fusionne des définitions partielles uniquement lorsque leurs chemins qualifiés complets sont identiques.

Recommandations

Les recommandations suivantes correspondent à une bonne approche en matière de programmation.

- **Noms uniques.** Lorsque vous contrôlez totalement l'attribution des noms aux éléments de programmation, il est toujours recommandé d'utiliser des noms uniques partout. Les noms identiques requièrent une qualification supplémentaire et peuvent compliquer la lecture de votre code. Ils sont également la source de petites erreurs et de résultats inattendus.
- **Qualification complète.** Lorsque vous travaillez avec des modules et d'autres éléments dans le même espace de noms, l'approche la plus sûre consiste à toujours utiliser la qualification complète pour tous les éléments de programmation. Si la promotion de type est invalidée pour un membre de module et que vous n'appliquez pas une qualification complète à ce membre, vous pouvez accéder par inadvertance à un élément de programmation différent.

Voir aussi

Tâches

Comment : contrôler la portée d'une variable

Référence

Module, instruction

Namespace, instruction

Partial (Visual Basic)

Concepts

Portée dans Visual Basic

Résolution d'une référence lorsque plusieurs variables ont le même nom

Niveaux d'accès dans Visual Basic

Le *niveau d'accès* d'un élément déclaré est l'étendue de la capacité à l'utiliser, à savoir le code qui est autorisé à lire l'élément ou à y écrire. Le niveau d'accès est déterminé d'une part par la façon dont vous déclarez l'élément lui-même et d'autre part par le niveau d'accès du conteneur de cet élément. Le code qui ne peut pas accéder à un élément conteneur ne peut accéder à aucun de ses éléments contenus, même qui sont ceux déclarés comme **Public**. Par exemple, une variable **Public** dans une structure **Private** est accessible à partir de l'intérieur de la classe contenant la structure, mais non à partir de l'extérieur de celle-ci.

Public

Le mot clé Public (Visual Basic) dans l'instruction de déclaration spécifie que les éléments sont accessibles à partir du code partout à l'intérieur d'un projet, à partir d'autres projets qui font référence à ce projet et à partir de tout assembly généré à partir de ce projet. Le code suivant illustre un exemple de déclaration **Public**.

```
Public Class classForEverybody
```

Vous pouvez utiliser le mot clé **Public** seulement au niveau du module, de l'interface ou de l'espace de noms. En d'autres termes, vous pouvez déclarer un élément public au niveau d'un fichier source ou d'un espace de noms, ou à l'intérieur d'une interface, d'un module, d'une classe ou d'une structure, mais pas dans une procédure.

Protected

Le mot clé Protected (Visual Basic) dans l'instruction de déclaration spécifique que les éléments sont accessibles uniquement à partir d'une même classe ou d'une classe dérivée de celle-ci. Le code suivant illustre un exemple de déclaration **Protected**.

```
Protected Class classForMyHeirs
```

Vous pouvez utiliser **Protected** uniquement au niveau de la classe et uniquement lors de la déclaration d'un membre d'une classe. Cela signifie que vous pouvez déclarer un élément protégé dans une classe, mais pas au niveau d'un fichier source ou d'un espace de noms, ni à l'intérieur d'une interface, d'un module, d'une structure ou d'une procédure.

Friend

Le mot clé Friend (Visual Basic) dans l'instruction de déclaration spécifie que les éléments sont accessibles à partir du même assembly, mais pas à partir de l'extérieur de l'assembly. Le code suivant illustre un exemple de déclaration **Friend**.

```
Friend stringForThisProject As String
```

Vous pouvez utiliser le mot clé **Friend** seulement au niveau du module, de l'interface ou de l'espace de noms. En d'autres termes, vous pouvez déclarer un élément "friend" au niveau d'un fichier source ou d'un espace de noms, ou à l'intérieur d'une interface, d'un module, d'une classe ou d'une structure, mais pas dans une procédure.

Protected Friend

Les mots clés **Protected** et **Friend** ensemble dans l'instruction de déclaration spécifient que les éléments sont accessibles soit à partir de classes dérivées, soit à partir du même assembly, soit les deux. Le code suivant illustre un exemple de déclaration **Protected Friend**.

```
Protected Friend stringForProjectAndHeirs As String
```

Vous pouvez utiliser **Protected Friend** uniquement au niveau de la classe et uniquement lors de la déclaration d'un membre d'une classe. Cela signifie que vous pouvez déclarer un élément "protected friend" dans une classe, mais pas au niveau d'un fichier source ou d'un espace de noms, ni à l'intérieur d'une interface, d'un module, d'une structure ou d'une procédure.

Private

Le mot clé Private (Visual Basic) dans l'instruction de déclaration spécifie que les éléments sont accessibles uniquement à partir du même module, de la même classe ou de la même structure. Le code suivant illustre un exemple de déclaration **Private**.

```
Private numberForMeOnly As Integer
```

Vous pouvez utiliser **Private** seulement au niveau du module. Cela signifie que vous pouvez déclarer un élément privé à l'intérieur d'un module, d'une classe ou d'une structure, mais pas au niveau d'un fichier source ou d'un espace de noms, ni à l'intérieur d'une interface ou d'une procédure.

Au niveau du module, l'instruction **Dim** utilisée sans mot clé de niveau d'accès équivaut à une déclaration **Private**. Toutefois, il peut s'avérer utile d'utiliser le mot clé **Private** afin que votre code soit plus facile à lire et à interpréter.

Modificateurs d'accès

Les mots clés qui spécifient le niveau d'accès portent le nom de *modificateurs d'accès*. Le tableau suivant compare les modificateurs d'accès.

Modificateur d'accès	**Niveau d'accès accordé**	**Éléments que vous pouvez déclarer avec ce niveau d'accès**	**Contexte de déclaration dans lequel vous pouvez utiliser ce modificateur**
Public	Illimité : Tout code qui peut voir un élément public peut y accéder	Interfaces Modules Classes Structures Membres de structures Procédures Propriétés Variables membres Constantes Énumérations Événements Déclarations externes Délégués	Fichier source Espace de noms Interface Module Class Structure
Protected	Dérivationnel : Le code situé dans la classe qui déclare un élément protégé, ou une classe dérivée d'elle, peut accéder à l'élément	Interfaces Classes Structures Procédures Propriétés Variables membres Constantes Énumérations Événements Déclarations externes Délégués	Class

Friend	Assembly : Le code situé dans l'assembly qui déclare un élément "friend " peut y accéder	Interfaces Modules Classes Structures Membres de structures Procédures Propriétés Variables membres Constantes Énumérations Événements Déclarations externes Délégués	Fichier source Espace de noms Interface Module Class Structure
Protected Friend	Union de **Protected** et **Friend** : Le code situé dans la même classe ou le même assembly qu' un élément "protected friend" ou dans toute classe dérivée de la classe de l'élément, peut y accéder	Interfaces Classes Structures Procédures Propriétés Variables membres Constantes Énumérations Événements Déclarations externes Délégués	Class
Private	Contexte de déclaration : Le code situé dans le type qui déclare un élément privé, y compris code situé dans des types contenus, peut accéder à l'élément	Interfaces Classes Structures Membres de structures Procédures Propriétés Variables membres Constantes Énumérations Événements Déclarations externes Délégués	Module Class Structure

Voir aussi

Tâches

Comment : contrôler la disponibilité d'une variable

Référence

Dim, instruction (Visual Basic)

Static (Visual Basic)

Concepts

Noms d'éléments déclarés

Caractéristiques d'éléments déclarés

Durée de vie dans Visual Basic

Portée dans Visual Basic

Variables en Visual Basic

Déclaration de variable dans Visual Basic

Autres ressources

Références aux éléments déclarés

Comment : contrôler la disponibilité d'une variable

Vous pouvez contrôler la disponibilité d'une variable en spécifiant son *niveau d'accès*. Le niveau d'accès détermine quel code a l'autorisation de lire ou d'écrire dans la variable.

- Les *variables membres* (définies au niveau du module et en dehors de toute procédure) autorisent par défaut un accès public, ce qui signifie que tout code peut les consulter et y accéder. Vous pouvez modifier cela en spécifiant un modificateur d'accès.
- Les *variables locales* (définies à l'intérieur d'une procédure) ont un accès public nominativement, bien que seul le code situé dans leur procédure puisse y accéder. Vous ne pouvez pas modifier le niveau d'accès d'une variable locale, mais vous pouvez modifier celui de la procédure qui la contient.

Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Accès privé et public

Pour rendre une variable accessible uniquement à partir de son module, de sa classe ou de sa structure

1. Placez l'Dim, instruction (Visual Basic) pour la variable à l'intérieur du module, de la classe ou de la structure, mais en dehors de toute procédure.
2. Incluez le mot clé Private (Visual Basic) dans l'instruction **Dim**.

   Vous pouvez lire ou écrire dans la variable à partir de tout emplacement dans le module, la classe ou la structure, mais pas depuis l'extérieur.

Pour rendre une variable accessible à partir de tout code qui peut le consulter

1. Pour une variable membre, placez l'instruction **Dim** pour la variable à l'intérieur d'un module, d'une classe ou d'une structure, mais en dehors de toute procédure.
2. Incluez le mot clé Public (Visual Basic) dans l'instruction **Dim**.

   Vous pouvez lire ou écrire dans la variable à partir de tout code qui interagit avec votre assembly.

- ou -

1. Pour une variable locale, placez l'instruction **Dim** pour la variable à l'intérieur d'une procédure.
2. N'incluez pas le mot clé **Public** dans l'instruction **Dim**.

   Vous pouvez lire ou écrire dans la variable à partir de tout emplacement dans la procédure, mais pas depuis l'extérieur.

Accès protégé et ami

Vous pouvez limiter le niveau d'accès d'une variable à sa classe et à toute classe dérivée, ou à son assembly. Vous pouvez spécifier également l'union de ces limitations, ce qui autorise l'accès à partir de code situé dans toute classe dérivée ou à tout autre emplacement dans le même assembly. Vous spécifiez cette union en combinant les mots clé **Protected** et **Friend** dans la même déclaration.

Pour rendre une variable accessible uniquement à partir de sa classe et de toute classe dérivée

1. Placez l'instruction **Dim** pour la variable à l'intérieur d'une classe, mais en dehors de toute procédure.
2. Incluez le mot clé Protected (Visual Basic) dans l'instruction **Dim**.

   Vous pouvez lire ou écrire dans la variable à partir de tout emplacement dans la classe, ainsi que depuis toute classe dérivée d'elle, mais pas depuis l'extérieur d'une classe dans la chaîne de dérivation.

Pour rendre une variable accessible uniquement à partir du même assembly

1. Placez l'instruction **Dim** pour la variable à l'intérieur d'un module, d'une classe ou d'une la structure, mais en dehors de toute procédure.

2. Incluez le mot clé Friend (Visual Basic) dans l'instruction **Dim**.

   Vous pouvez lire ou écrire dans la variable à partir de tout emplacement dans le module, la classe ou la structure, ainsi qu'à partir de tout code situé dans le même assembly, mais pas depuis l'extérieur de l'assembly.

Exemple

L'exemple suivant montre des déclarations de variables avec des niveaux d'accès **Public**, **Protected**, **Friend**, **Protected Friend** et **Private**. Notez que lorsque l'instruction **Dim** spécifie un niveau d'accès, il n'est pas nécessaire d'inclure le mot clé **Dim**.

```
Public Class classForEverybody
Protected Class classForMyHeirs
Friend stringForThisProject As String
Protected Friend stringForProjectAndHeirs As String
Private numberForMeOnly As Integer
```

Sécurité

Plus le niveau d'accès d'une variable est restrictif, moins il y a de risques que du code malveillant en fasse une utilisation incorrecte.

Voir aussi

Référence

Dim, instruction (Visual Basic)
Public (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)
Private (Visual Basic)

Concepts

Niveaux d'accès dans Visual Basic

Références aux éléments déclarés

Lorsque votre code fait référence à un élément déclaré, le compilateur Visual Basic établit une correspondance entre le nom indiqué dans votre référence et la déclaration appropriée de ce nom. Si plusieurs éléments sont déclarés avec le même nom, vous pouvez contrôler lequel de ces éléments doit être référencé en *qualifiant* son nom.

Dans cette section

Résolution d'une référence lorsque plusieurs variables ont le même nom

Explique comment le compilateur fait correspondre une référence à une déclaration.

Comment : qualifier un nom d'élément déclaré

Explique comment qualifier un nom d'élément pour faciliter une correspondance de référence.

Comment : faire la distinction entre deux éléments portant le même nom

Décrit comment qualifier un nom d'élément lors de son appel pour permettre au compilateur Visual Basic d'établir une correspondance entre votre référence et l'élément souhaité.

Occultation dans Visual Basic

Décrit le traitement des éléments portant le même nom.

Rubriques connexes

Noms d'éléments déclarés

Explique comment nommer les éléments et utiliser la casse alphabétique.

Caractéristiques d'éléments déclarés

Présente les caractéristiques telles que la portée, que possèdent les éléments déclarés.

Résolution d'une référence lorsque plusieurs variables ont le même nom

Lorsque le compilateur tente de faire correspondre une référence de nom à une déclaration de nom, il recherche la déclaration correspondante avec la *plus petite portée*. Cela signifie qu'il commence par le code qui effectue la référence et qu'il parcourt ensuite les niveaux successifs d'éléments conteneurs.

Si vous souhaitez substituer ce processus de recherche et spécifier un nom déclaré dans une portée plus générale, vous devez *qualifier* le nom avec l'élément conteneur de la portée plus générale. Dans certains cas, vous devrez peut-être qualifier aussi l'élément conteneur. Pour plus d'informations sur la qualification de noms, consultez Comment : qualifier un nom d'élément déclaré.

Vous devrez peut-être également qualifier une référence de nom si votre application a accès à plusieurs éléments de programmation avec le même nom. Pour obtenir des exemples, consultez « Classes avec le même nom » dans cette page d'aide et Comment : faire la distinction entre deux éléments portant le même nom.

La plus petite portée

L'exemple suivant montre des références à deux variables avec le même nom.

```
' Assume these two modules are both in the same assembly.
Module container
    Public totalCount As Integer = 1
    Public Sub showCount()
        Dim totalCount As Integer = 6000
        ' The following statement displays the local totalCount (6000).
        MsgBox("Unqualified totalCount is " & CStr(totalCount))
        ' The following statement displays the module's totalCount (1).
        MsgBox("container.totalCount is " & CStr(container.totalCount))
    End Sub
End Module
Module callingModule
    Public Sub displayCount()
        container.showCount()
        ' The following statement displays the containing module's totalCount (1).
        MsgBox("container.totalCount is " & CStr(container.totalCount))
    End Sub
End Module
```

L'exemple précédent déclare deux variables, chacune nommée `totalCount`, à différents niveaux de portée dans le module `container`. Lorsque la procédure `showCount` affiche `totalCount` sans qualification, le compilateur Visual Basic résout la référence à la déclaration avec la plus petite portée, à savoir la déclaration locale à l'intérieur de `showCount`. Lorsqu'il qualifie `totalCount` avec le module conteneur `container`, le compilateur résout la référence à la déclaration avec la portée plus générale.

Membres d'autres éléments conteneurs

Lorsque vous utilisez un membre non partagé d'une autre classe ou structure, vous devez d'abord qualifier le nom du membre avec une variable ou une expression qui pointe vers une instance de la classe ou structure. Dans l'exemple suivant, `demoClass` est une instance d'une classe nommée `class1`.

```
Dim demoClass As class1 = New class1()
demoClass.someSub[(argumentlist)]
```

Vous ne pouvez pas utiliser le nom de la classe pour qualifier un membre qui n'est pas Shared (Visual Basic). Vous devez d'abord créer une instance dans une variable objet (dans ce cas-ci `demoClass`), puis la référencer par le nom de la variable.

Si une classe ou une structure a un membre **Shared**, vous pouvez qualifier ce membre avec le nom de structure ou de classe ou avec une variable ou expression qui pointe vers une instance.

Un module n'a pas d'instances séparées et tous ses membres sont **Shared** par défaut. Par conséquent, vous qualifiez un membre de module avec le nom du module.

Exemple de référence qualifiée

L'exemple suivant montre des références qualifiées à des procédures de membre de module.

```
' Assume these three modules are all in the same assembly.
Module module1
    Public Sub perform()
        MsgBox("module1.perform() now returning")
    End Sub
End Module
Module module2
    Public Sub perform()
        MsgBox("module2.perform() now returning")
    End Sub
    Public Sub doSomething()
        ' The following statement calls perform in module2, the active module.
        perform()
        ' The following statement calls perform in module1.
        module1.perform()
    End Sub
End Module
Module module3
    Public Sub callPerform()
        ' The following statement calls perform in module1.
        module1.perform()
        ' The following statement makes an unresolvable name reference
        ' and therefore generates a COMPILER ERROR.
        perform() ' INVALID statement
    End Sub
End Module
```

L'exemple précédent déclare deux procédures **Sub**, toutes deux nommées `perform`, dans des modules différents d'un projet. Chacune d'elles peut être spécifiée sans qualification dans son propre module, mais doit être qualifiée si elle est référencée ailleurs. La dernière référence dans `module3` ne qualifiant pas `perform`, le compilateur est incapable de résoudre cette référence.

Références aux projets

Pour utiliser des éléments Public (Visual Basic) définis dans un autre projet, vous devez avant tout définir une *référence* à l'assembly ou la bibliothèque de types de ce projet. Vous pouvez définir une référence en sélectionnant **Ajouter une référence** dans le menu **Projet**, ou avec l'option du compilateur de ligne de commande /reference (Visual Basic).

Vous pouvez, par exemple, utiliser le modèle objet XML du .NET Framework. Si vous définissez une référence à l'espace de noms System.Xml, vous pouvez déclarer et utiliser n'importe laquelle de ses classes, telles que XmlDocument. L'exemple suivant utilise **XmlDocument** :

```
' Assume this project has a reference to System.Xml
' The following statement creates xDoc as an XML document object.
Dim xDoc As System.Xml.XmlDocument
```

Importation d'éléments conteneurs

Il est possible d'utiliser l'instruction Imports pour *importer* les espaces de noms contenant les modules ou classes que vous souhaitez utiliser. Cela vous permet de faire référence aux éléments définis dans un espace de noms importé sans qualifier complètement leurs noms. L'exemple suivant réécrit l'exemple précédent pour importer l'espace de noms **System.Xml**.

```
' Assume this project has a reference to System.Xml
' The following statement must precede all your declarations.
Imports System.Xml
' The following statement creates xDoc as an XML document object.
Dim xDoc As XmlDocument
```

En outre, l'instruction **Imports** peut définir un *alias d'importation* pour chaque espace de noms importé. Ainsi, votre code source est plus court et plus facile à lire. L'exemple suivant réécrit l'exemple précédent pour utiliser `xD` en tant qu'alias pour l'espace de noms **System.Xml**.

```
' Assume this project has a reference to System.Xml
' The following statement must precede all your declarations.
Imports xD = System.Xml
' The following statement creates xDoc as an XML document object.
Dim xDoc As xD.XmlDocument
```

L'instruction **Imports** ne rend pas d'éléments d'autres projets disponibles à votre application. C'est-à-dire qu'elle ne prend pas la place de la définition d'une référence. L'importation d'un espace de noms rend simplement inutile la qualification des noms définis dans cet espace de noms.

Vous pouvez aussi utiliser l'instruction **Imports** pour importer des modules, des classes, des structures et des énumérations. Vous pouvez ensuite utiliser les membres de ces éléments importés sans qualification. Toutefois, vous devez toujours qualifier les membres non partagés de classes et de structures avec une variable ou expression qui correspond à une instance de la classe ou de la structure.

Classes portant le même nom

Lorsque vous créez une nouvelle instance d'un objet, vous devrez peut-être qualifier la classe avec l'espace de noms ou la bibliothèque de types à laquelle elle appartient. Par exemple, les espaces de noms System.Windows.Forms et System.Web.UI.WebControls contiennent tous deux une classe `Label` (System.Windows.Forms.Label et System.Web.UI.WebControls.Label). Si votre application utilise les deux ou si elle définit sa propre classe `Label`, vous devez faire la distinction entre les différents objets `Label`. Incluez l'alias d'espace de noms ou d'importation dans la déclaration de variable. L'exemple suivant utilise l'alias d'importation.

```
' The following statement must precede all your declarations.
Imports win = System.Windows.Forms, web = System.Web.UI.WebControls
' The following statement references the Windows.Forms.Label class.
Dim winLabel As New win.Label()
```

Indications concernant l'attribution d'un nom

Lorsque vous définissez deux ou plusieurs éléments de programmation avec le même nom, il peut en résulter une *ambiguïté de noms* lorsque le compilateur tente de résoudre une référence à ce nom. Si plusieurs définitions sont dans la portée ou si aucune définition n'est dans la portée, la référence ne peut pas être résolue. Pour obtenir un exemple, consultez « Exemple de référence qualifiée » dans cette page d'aide.

Pour éviter toute ambiguïté, vous pouvez attribuer des noms uniques à tous vos éléments. Ensuite, vous pouvez faire référence à tout élément sans devoir qualifier son nom avec un espace de noms, un module ou une classe. Vous réduisez également les risques de faire référence par erreur à l'élément incorrect.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration
Comment : qualifier un nom d'élément déclaré
Comment : faire la distinction entre deux éléments portant le même nom

Référence

Imports, instruction
New (Visual Basic)
Public (Visual Basic)

Concepts

Variables en Visual Basic

Autres ressources

Références aux éléments déclarés

Comment : qualifier un nom d'élément déclaré

Lorsque votre code fait référence à un élément déclaré, le compilateur Visual Basic doit établir une correspondance entre le nom indiqué dans votre référence et la déclaration appropriée de ce nom. Si l'élément est défini à l'extérieur de votre code ou si plusieurs éléments sont déclarés avec le même nom, vous devrez peut-être *qualifier* le nom pour permettre au compilateur de résoudre toute ambiguïté possible ou de localiser l'élément.

La qualification d'un nom revient à le faire précéder, dans votre instruction source, d'informations qui identifient l'emplacement où l'élément cible est défini. Ces informations portent le nom de *chaîne de qualification*. Elles peuvent inclure un ou plusieurs espaces de noms et un module, une classe ou une structure.

La chaîne de qualification doit spécifier clairement le module, la classe ou la structure qui contient l'élément cible. Le conteneur peut se trouver dans un autre élément conteneur, en général un espace de noms. Vous devrez peut-être inclure plusieurs éléments conteneurs dans la chaîne de qualification.

Pour accéder à un élément déclaré en qualifiant son nom

1. Déterminez l'emplacement dans lequel l'élément a été défini. Il peut s'agir d'un espace de noms, ou même d'une hiérarchie d'espaces de noms. Dans l'espace de noms de plus bas niveau, l'élément doit être contenu dans un module, une classe ou une structure.

   ```
   ' Assume the following hierarchy exists outside your code.
   Namespace outerSpace
       Namespace innerSpace
           Module holdsTotals
               Public Structure totals
                   Public thisTotal As Integer
                   Public Shared grandTotal As Long
               End Structure
           End Module
       End Namespace
   End Namespace
   ```

2. Déterminez un chemin d'accès de qualification basé sur l'emplacement de l'élément cible. Commencez avec l'espace de noms de plus haut niveau, continuez avec l'espace de noms de plus bas niveau, puis finissez par le module, la classe ou la structure qui contient l'élément cible. Chaque élément du chemin d'accès doit contenir l'élément qui le suit.

   ```
   outerSpace innerSpace holdsTotals totals
   ```

3. Préparez la chaîne de qualification pour l'élément cible. Placez un point (**.**) après chaque élément dans le chemin d'accès. Votre application doit avoir accès à chaque élément de votre chaîne de qualification.

   ```
   outerSpace.innerSpace.holdsTotals.totals.
   ```

4. Écrivez de manière normale l'instruction d'expression ou d'assignation qui fait référence à l'élément cible.

   ```
   grandTotal = 9000
   ```

5. Placez la chaîne de qualification avant le nom de l'élément cible. Le nom doit suivre immédiatement le point (**.**) qui suit le module, la classe ou la structure qui contient l'élément.

   ```
   ' Assume the following module is part of your code.
   Module accessGrandTotal
       Public Sub setGrandTotal()
           outerSpace.innerSpace.holdsTotals.totals.grandTotal = 9000
       End Sub
   End Module
   ```

6. Le compilateur utilise la chaîne de qualification pour rechercher une déclaration claire et non équivoque à laquelle il peut faire correspondre la référence de l'élément cible.

Voir aussi

Tâches

Comment : faire la distinction entre deux éléments portant le même nom
Comment : masquer une variable portant le même nom que votre variable
Comment : masquer une variable héritée
Comment : accéder à une variable masquée par une classe dérivée

Concepts

Caractéristiques d'éléments déclarés
Résolution d'une référence lorsque plusieurs variables ont le même nom
Instructions de déclaration en Visual Basic

Autres ressources

Références aux éléments déclarés

Comment : faire la distinction entre deux éléments portant le même nom

Si votre application a accès à plusieurs éléments portant le même nom, vous pouvez *qualifier* un nom pour permettre au compilateur Visual Basic de faire correspondre votre référence à l'élément particulier que vous souhaitez utiliser. Pour plus d'informations, consultez Comment : qualifier un nom d'élément déclaré.

Exemple

L'exemple suivant montre deux variables portant le même nom situées dans différents éléments contenants du même nom. Les variables sont déclarées comme **Shared** simplement pour raccourcir le code dans l'exemple.

```
Namespace space1
    Public Class innerClass
        ' String showMe is declared Shared to facilitate reference.
        Public Shared showMe As String = "Shared string 1"
    End Class
End Namespace
Namespace space2
    Public Class innerClass
        ' String showMe is declared Shared to facilitate reference.
        Public Shared showMe As String = "Shared string 2"
    End Class
End Namespace
Public Module callShowMe
    Public Sub showStrings()
        MsgBox("From space1: " & space1.innerClass.showMe _
            & vbCrLf & "From space2: " & space2.innerClass.showMe)
    End Sub
End Module
```

L'exemple précédent déclare deux variables, chacune nommée `showMe` et chacune située à l'intérieur d'une classe nommée `innerClass`. Les deux éléments contenants ayant le même nom, le code appelant doit non seulement qualifier la variable `showMe` avec son conteneur `innerClass`, mais également qualifier `innerClass` avec son conteneur `space1` ou `space2`. Le compilateur Visual Basic peut résoudre chaque référence, car chaque chaîne de qualification fournit un chemin d'accès non équivoque à une déclaration unique.

Programmation fiable

Moins vous avez déclaré de variables avec le même nom, moins vous risquez de faire référence par erreur à l'une plutôt qu'à une autre. Vous pouvez également réduire les problèmes de correspondance de référence.

Sécurité

Moins vous avez déclaré de variables avec le même nom, moins vous risquez que du code malveillant puisse les utiliser de manière incorrecte.

Voir aussi

Tâches

Comment : qualifier un nom d'élément déclaré
Comment : masquer une variable portant le même nom que votre variable
Comment : masquer une variable héritée
Comment : accéder à une variable masquée par une classe dérivée

Concepts

Caractéristiques d'éléments déclarés
Résolution d'une référence lorsque plusieurs variables ont le même nom
Instructions de déclaration en Visual Basic

Autres ressources

Références aux éléments déclarés

Occultation dans Visual Basic

Lorsque deux éléments de programmation partagent le même nom, l'un des deux peut masquer (ou *occulter*) l'autre. Dans ce cas, l'élément occulté n'est pas disponible à des fins de référence ; en effet, lorsque votre code utilise le nom de l'élément, le compilateur Visual Basic le résout en l'élément occultant.

But

L'objectif principal de l'occultation est de protéger la définition de vos membres de classe. La classe de base peut subir un changement qui crée un élément du même nom que celui que vous avez déjà défini. Dans ce cas, le modificateur **Shadows** oblige les références de votre classe à être résolues vers le membre que vous avez défini, et non vers le nouvel élément de la classe de base.

Types d'occultation

Un élément peut occulter un autre élément de deux façons différentes. L'élément occultant peut être déclaré dans une sous-région de la région contenant l'élément occulté ; dans ce cas, l'occultation s'effectue *par l'intermédiaire de la portée*. Une classe dérivante peut également redéfinir un membre d'une classe de base ; dans ce cas, l'occultation se produit *par l'intermédiaire de l'héritage*.

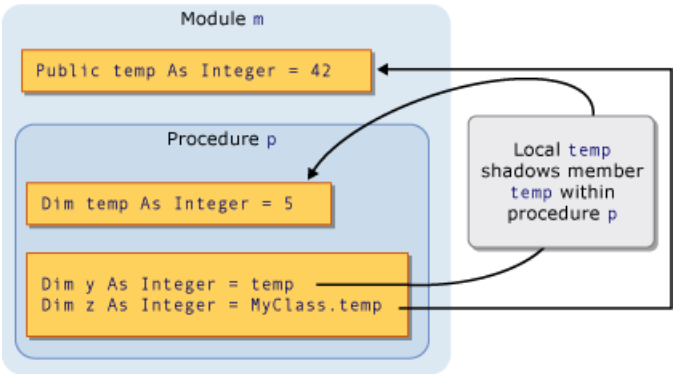
Occultation par l'intermédiaire de la portée

Des éléments de programmation du même module, de la même classe ou de la même structure peuvent avoir le même nom, mais une portée différente. Lorsque deux éléments sont déclarés de cette façon et que le code fait référence au nom qu'ils partagent, l'élément à la portée la plus restreinte occulte l'autre élément (la portée de bloc est la plus restreinte).

Ainsi, un module peut définir une variable **Public** appelée `temp` et une procédure du module peut déclarer une variable locale également appelée `temp`. Les références à `temp` à partir de la procédure accèdent à la variable locale, tandis que les références à `temp` à partir de l'extérieur de la procédure accèdent à la variable **Public**. Dans ce cas, la variable `temp` de la procédure occulte la variable `temp` du module.

L'illustration suivante montre deux variables, toutes deux nommées `temp`. La variable locale `temp` occulte la variable membre `temp` lorsque l'utilisateur y accède à partir de sa propre procédure `p`. Toutefois, le mot clé **MyClass** outrepasse l'occultation et accède à la variable membre.

Occultation par l'intermédiaire de la portée



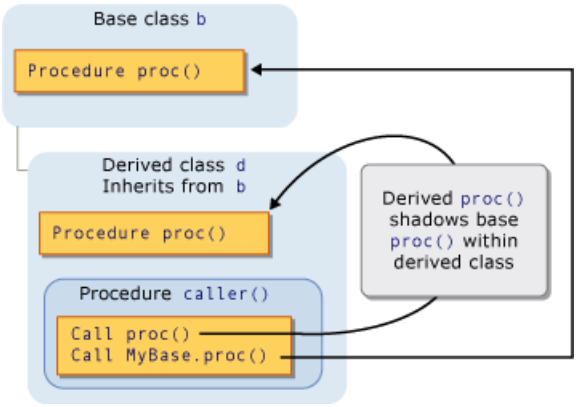
Pour obtenir un exemple d'occultation par l'intermédiaire de la portée, consultez Comment : masquer une variable portant le même nom que votre variable.

Occultation par héritage

Si une classe dérivée redéfinit un élément de programmation hérité d'une classe de base, l'élément qui redéfinit occulte l'élément d'origine. Vous pouvez occulter n'importe quel type d'élément déclaré, ou ensemble d'éléments surchargés, avec un autre type. Par exemple, une variable **Integer** peut occulter une procédure **Function**. Si vous occultez une procédure par une autre procédure, vous pouvez utiliser une liste de paramètres différente et un type de retour différent.

L'illustration suivante montre une classe de base `b` et une classe dérivée `d` qui hérite de `b`. La classe de base définit une procédure nommée `proc` et la classe dérivée l'occulte avec une autre procédure du même nom. La première instruction **Call** accède au `proc` occultant dans la classe dérivée. Toutefois, le mot clé **MyBase** outrepasse l'occultation et accède à la procédure occultée dans la classe de base.

Occultation par héritage



Pour obtenir un exemple d'occultation par l'intermédiaire de l'héritage, consultez
Comment : masquer une variable portant le même nom que votre variable et Comment : masquer une variable héritée.

Occultation et niveau d'accès

L'élément occultant n'est pas toujours accessible à partir du code à l'aide de la classe dérivée. Par exemple, il peut être déclaré **Private**. Dans ce cas, l'occultation échoue et le compilateur résout toute référence au même élément qu'il aurait en l'absence d'occultation. Il s'agit de l'élément accessible qui présente le moins d'étapes de dérivation vers l'arrière à partir de la classe d'occultation. Si l'élément occulté correspond à une procédure, la résolution est effectuée vers la version la plus proche accessible dotée des mêmes nom, liste de paramètres et type de retour.

L'exemple suivant montre une hiérarchie d'héritage de trois classes. Chaque classe définit une procédure **Sub** `display` et chaque classe dérivée occulte la procédure `display` dans sa classe de base.

```
Public Class firstClass
    Public Sub display()
        MsgBox("This is firstClass")
    End Sub
End Class
Public Class secondClass
    Inherits firstClass
    Private Shadows Sub display()
        MsgBox("This is secondClass")
    End Sub
End Class
Public Class thirdClass
    Inherits secondClass
    Public Shadows Sub display()
        MsgBox("This is thirdClass")
    End Sub
End Class
Module callDisplay
    Dim first As New firstClass
    Dim second As New secondClass
    Dim third As New thirdClass
    Public Sub callDisplayProcedures()
        ' The following statement displays "This is firstClass".
        first.display()
        ' The following statement displays "This is firstClass".
        second.display()
        ' The following statement displays "This is thirdClass".
        third.display()
    End Sub
End Module
```

Dans l'exemple précédent, la classe dérivée `secondClass` occulte `display` avec une procédure **Private**. Lorsque le module `callDisplay` appelle `display` dans `secondClass`, le code appelant est à l'extérieur de `secondClass` et par conséquent ne peut pas accéder à la procédure `display` privée. L'occultation échoue et le compilateur résout la référence vers la procédure `display` de la classe de base.

Toutefois, la classe dérivée supplémentaire `thirdClass` déclare `display` comme **Public** ; par conséquent, le code dans `callDisplay` peut y accéder.

Occultation et substitution

Ne confondez pas l'occultation et la substitution. Elles sont toutes deux utilisées lorsqu'une classe dérivée hérite d'une classe de base et elles redéfinissent un élément déclaré par un autre. Il existe toutefois des différences notables entre l'occultation et la substitution. Pour obtenir une comparaison, consultez Différences entre l'occultation et la substitution.

Occultation et surcharge

Si vous occultez le même élément de classe de base avec plusieurs éléments de votre classe dérivée, les éléments occultants deviennent des versions surchargées de cet élément. Pour plus d'informations, consultez Surcharge de procédure.

Accès à un élément occulté

Lorsque vous accédez à un élément à partir d'une classe dérivée, vous le faites normalement par l'instance en cours de cette classe dérivée, en qualifiant le nom de l'élément avec le mot clé Me. Si votre classe dérivée occulte l'élément figurant dans la classe de base, vous pouvez accéder à l'élément de la classe de base en le qualifiant avec le mot clé MyBase.

Pour obtenir un exemple d'accès à un élément occulté, consultez
Comment : accéder à une variable masquée par une classe dérivée.

Déclaration de la variable objet

La façon dont vous créez la variable objet peut également déterminer si la classe dérivée accède à un élément occultant ou à l'élément occulté. L'exemple suivant crée deux objets à partir d'une classe dérivée, mais l'un des objets est déclaré comme classe de base et l'autre comme classe dérivée.

```
Public Class baseCls
    ' The following statement declares the element that is to be shadowed.
    Public z As Integer = 100
End Class
Public Class dervCls
    Inherits baseCls
    ' The following statement declares the shadowing element.
    Public Shadows z As String = "*"
End Class
Public Class useClasses
    ' The following statement creates the object declared as the base class.
    Dim basObj As baseCls = New dervCls()
    ' Note that dervCls widens to its base class baseCls.
    ' The following statement creates the object declared as the derived class.
    Dim derObj As dervCls = New dervCls()
    Public Sub showZ()
    ' The following statement outputs 100 (the shadowed element).
        MsgBox("Accessed through base class: " & basObj.z)
    ' The following statement outputs "*" (the shadowing element).
        MsgBox("Accessed through derived class: " & derObj.z)
    End Sub
End Class
```

Dans l'exemple précédent, la variable `basObj` est déclarée comme la classe de base. L'assignation d'un objet `dervCls` à cette classe constitue une conversion étendue et est donc valide. Toutefois, la classe de base ne pouvant pas accéder à la version occultante de la variable `z` dans la classe dérivée, le compilateur résout `basObj.z` à la valeur de la classe de base d'origine.

Voir aussi

Référence
Shadows
Overrides
Me
MyBase
Concepts
Portée dans Visual Basic
Conversions étendues et restrictives
Autres ressources
Références aux éléments déclarés

Différences entre l'occultation et la substitution

Lorsque vous définissez une classe qui hérite d'une classe de base, vous souhaitez parfois redéfinir un ou plusieurs des éléments de classe de base dans la classe dérivée. L'occultation et la substitution sont disponibles à cette fin.

Comparaison

Il est facile de confondre occulter et substituer. Elles sont toutes deux utilisées lorsqu'une classe dérivée hérite d'une classe de base et elles redéfinissent un élément déclaré par un autre. Il existe toutefois des différences importantes entre l'occultation et la substitution.

Le tableau suivant compare l'occultation à la substitution.

Point de comparaison	Occultation	Substitution
But	Protéger contre une modification de classe de base suivante qui introduit un membre déjà défini dans votre classe dérivée	Atteindre le polymorphisme en définissant une implémentation différente d'une procédure ou d'une propriété avec la même séquence d'appel [1]
Élément redéfini	Tout type d'élément déclaré	Seulement une procédure (**Function**, **Sub** ou **Operator**) ou une propriété
Élément redéfinissant	Tout type d'élément déclaré	Seulement une procédure ou une propriété avec une séquence d'appel identique[1]
Niveau d'accès de l'élément redéfinissant	N'importe quel niveau d'accès	Impossible de modifier le niveau d'accès de l'élément substitué
Lisibilité et accessibilité en écriture de l'élément redéfinissant	Toute combinaison	Ne peut pas modifier la lisibilité et la facilité d'écriture de la propriété substituée
Contrôle sur la redéfinition	L'élément de classe de base ne peut pas appliquer ou interdire l'occultation	L'élément de classe de base peut spécifier **MustOverride**, **NotOverridable** ou **Overridable**
Utilisation de mot clé	**Shadows** recommandé dans la classe dérivée ; **Shadows** supposé lorsque ni **Shadows** ni **Overrides** ne sont spécifiés[2]	**Overridable** ou **MustOverride** requis dans la classe de base ; **Overrides** requis dans la classe dérivée
Héritage d'élément redéfinissant par des classes dérivant de votre classe dérivée	Élément occultant hérité par des classes plus dérivées ; élément occulté toujours masqué[3]	Élément substituant hérité par des classes plus dérivées ; élément substitué toujours substitué

[1] La *séquence d'appel* se compose du type d'élément (**Function**, **Sub**, **Operator** ou **Property**), du nom, de la liste d'arguments et du type de retour. Vous ne pouvez pas substituer une procédure par une propriété, ou vice versa. Vous ne pouvez pas substituer un genre de procédure (**Function**, **Sub** ou **Operator**) par un autre genre.

[2] Si vous ne spécifiez pas **Shadows** ou **Overrides**, le compilateur signale un message d'avertissement pour vous aider à vérifier le genre de redéfinition que vous souhaitez utiliser. Si vous ignorez l'avertissement, le mécanisme d'occultation est utilisé.

[3] Si l'élément occultant est inaccessible dans une classe plus dérivée, l'occultation n'est pas héritée. Par exemple, si vous déclarez l'élément occultant comme **Private**, une classe dérivant de votre classe dérivée hérite de l'élément d'origine et non de l'élément occultant.

Indications

Vous utilisez normalement la substitution dans les cas suivants :

- Vous définissez des classes dérivées polymorphes.

- Vous souhaitez la sécurité du compilateur qui applique le type d'élément et la séquence d'appel identiques.

Vous utilisez normalement l'occultation dans les cas suivants :

- Vous savez que votre classe de base peut être modifiée et peut définir un élément à l'aide du même nom que le vôtre.
- Vous souhaitez pouvoir modifier le type d'élément ou la séquence d'appel.

Voir aussi

Tâches

Comment : faire la distinction entre deux éléments portant le même nom
Comment : masquer une variable portant le même nom que votre variable
Comment : masquer une variable héritée
Comment : accéder à une variable masquée par une classe dérivée

Référence

Shadows
Overrides

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom
Occultation dans Visual Basic

Autres ressources

Références aux éléments déclarés

Comment : masquer une variable portant le même nom que votre variable

Vous pouvez masquer une variable en l'*occultant*, c'est-à-dire en la redéfinissant avec une variable du même nom. Vous pouvez occulter la variable à masquer de deux manières :

- **Occultation par l'intermédiaire de la portée.** Vous pouvez l'occulter par l'intermédiaire de la portée en la redéclarant dans une sous-région de la région contenant la variable à masquer.
- **Occultation par héritage.** Si la variable à masquer est définie au niveau de la classe, vous pouvez l'occulter par l'intermédiaire de l'héritage en la rédeclarant avec le mot clé Shadows dans une classe dérivée.

Deux manières de masquer une variable

Pour masquer une variable en l'occultant par l'intermédiaire de la portée

1. Déterminez la région qui définit la variable à masquer et déterminez une sous-région dans laquelle la redéfinir avec votre variable.

Région de variable	Sous-région autorisée pour la redéfinir
Module	Classe dans le module
Classe	Sous-classe dans la classe Procédure dans la classe

   Vous ne pouvez pas redéfinir de variable de procédure dans un bloc au sein de cette procédure, par exemple dans une construction **If**...**End If** ou une boucle **For**.

2. Créez la sous-région si elle n'existe pas déjà.

3. Dans la sous-région, écrivez à une Dim, instruction (Visual Basic) qui déclare la variable occultante.

   Lorsque le code à l'intérieur de la sous-région fait référence au nom de variable, le compilateur résout la référence à la variable occultante.

   L'exemple suivant illustre l'occultation par l'intermédiaire de la portée, ainsi qu'une référence qui outrepasse l'occultation.

   ```
   Module shadowByScope
       ' The following statement declares num as a module-level variable.
       Public num As Integer
       Sub show()
           ' The following statement declares num as a local variable.
           Dim num As Integer
           ' The following statement sets the value of the local variable.
           num = 2
           ' The following statement displays the module-level variable.
           MsgBox(CStr(shadowByScope.num))
       End Sub
       Sub useModuleLevelNum()
           ' The following statement sets the value of the module-level variable.
           num = 1
           show()
       End Sub
   End Module
   ```

   L'exemple précédent déclare la variable `num` à la fois au niveau du module et au niveau de la procédure (dans la procédure `show`). La variable locale `num` occulte la variable de niveau module `num` dans `show`, par conséquent la variable

locale a la valeur 2. Toutefois, il n'existe aucune variable locale pour occulter `num` dans la procédure `useModuleLevelNum`. Par conséquent, `useModuleLevelNum` définit la valeur de la variable au niveau du module à 1.

L'appel **MsgBox** à l'intérieur de `show` outrepasse le mécanisme d'occultation en qualifiant `num` avec le nom du module. Par conséquent, il affiche la variable au niveau du module au lieu de la variable locale.

Pour masquer une variable en l'occultant par l'intermédiaire de l'héritage

1. Assurez-vous que la variable à masquer est déclarée dans une classe et au niveau de la classe (en dehors de toute procédure). Sinon, vous ne pouvez pas l'occulter par héritage.
2. Définissez une classe dérivée de la classe de la variable s'il n'en n'existe pas déjà une.
3. À l'intérieur de la classe dérivée, écrivez une instruction **Dim** qui déclare votre variable. Incluez le mot clé Shadows dans la déclaration.

   Lorsque le code de la classe dérivée fait référence au nom de la variable, le compilateur résout la référence à votre variable.

   L'exemple suivant illustre l'occultation par l'intermédiaire de l'héritage. Il effectue deux références, une qui accède à la variable occultante et une qui outrepasse l'occultation.

```
Public Class shadowBaseClass
    Public shadowString As String = "This is the base class string."
End Class
Public Class shadowDerivedClass
    Inherits shadowBaseClass
    Public Shadows shadowString As String = "This is the derived class string."
    Public Sub showStrings()
        Dim s As String = "Unqualified shadowString: " & shadowString _
            & vbCrLf & "MyBase.shadowString: " & MyBase.shadowString
        MsgBox(s)
    End Sub
End Class
```

   L'exemple précédent déclare la variable `shadowString` dans la classe de base et la masque dans la classe dérivée. La procédure `showStrings` dans la classe dérivée affiche la version occultante de la chaîne lorsque le nom `shadowString` n'est pas qualifié. Elle affiche ensuite la version occultée lorsque `shadowString` est qualifié avec le mot clé MyBase.

Programmation fiable

L'occultation introduit plusieurs versions d'une variable avec le même nom. Lorsqu'une instruction de code fait référence au nom de la variable, la version vers laquelle le compilateur résout la référence dépend de plusieurs facteurs, tels que l'emplacement de l'instruction de code et la présence d'une chaîne qualifiante. Cela peut augmenter le risque de référence à une version non souhaitée d'une variable occultée. Vous pouvez réduire ce risque en qualifiant complètement toutes les références à une variable occultée.

Voir aussi

Tâches

Comment : faire la distinction entre deux éléments portant le même nom
Comment : masquer une variable héritée
Comment : accéder à une variable masquée par une classe dérivée

Référence

Overrides
MyBase

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom
Occultation dans Visual Basic
Différences entre l'occultation et la substitution

Autres ressources

Références aux éléments déclarés

Comment : masquer une variable héritée

Une classe dérivée hérite de toutes les définitions de sa classe de base. Si vous souhaitez définir une variable à l'aide du même nom qu'un élément de la classe de base, vous pouvez masquer (ou *occulter*) cet élément de classe de base lorsque vous définissez votre variable dans la classe dérivée. Si vous procédez ainsi, le code de la classe dérivée accède à votre variable à moins qu'il n'outrepasse de manière explicite le mécanisme occultant.

L'une des autres raisons pour lesquelles vous pourriez souhaiter masquer une variable héritée serait pour la protéger de la révision de classe de base. La classe de base peut subir une modification qui altère l'élément dont vous héritez. Si cela se produit, le modificateur **Shadows** force les références de la classe dérivée à être résolues vers votre variable plutôt que vers l'élément de classe de base.

Pour masquer une variable héritée

1. Assurez-vous que la variable à masquer est déclarée au niveau de la classe (en dehors de toute procédure). Autrement, il n'est pas nécessaire de la masquer.
2. À l'intérieur de votre classe dérivée, écrivez une Dim, instruction (Visual Basic) qui déclare votre variable. Utilisez le même nom que celui de la variable héritée.
3. Incluez le mot clé Shadows dans la déclaration.

   Lorsque le code de la classe dérivée fait référence au nom de la variable, le compilateur résout la référence vers votre variable.

   L'exemple suivant illustre l'occultation d'une variable héritée.

```
Public Class shadowBaseClass
    Public shadowString As String = "This is the base class string."
End Class
Public Class shadowDerivedClass
    Inherits shadowBaseClass
    Public Shadows shadowString As String = "This is the derived class string."
    Public Sub showStrings()
        Dim s As String = "Unqualified shadowString: " & shadowString _
            & vbCrLf & "MyBase.shadowString: " & MyBase.shadowString
        MsgBox(s)
    End Sub
End Class
```

   L'exemple précédent déclare la variable `shadowString` dans la classe de base et la masque dans la classe dérivée. La procédure `showStrings` dans la classe dérivée affiche la version occultante de la chaîne lorsque le nom `shadowString` n'est pas qualifié. Elle affiche ensuite la version occultée lorsque `shadowString` est qualifié avec le mot clé MyBase.

Programmation fiable

L'occultation introduit plusieurs versions d'une variable avec le même nom. Lorsqu'une instruction de code fait référence au nom de la variable, la version vers laquelle le compilateur résout la référence dépend de plusieurs facteurs, tels que l'emplacement de l'instruction de code et la présence d'une chaîne qualifiante. Cela peut augmenter le risque de référence à une version non souhaitée d'une variable occultée. Vous pouvez réduire ce risque en effectuant une qualification complète de toutes les références vers une variable occultée.

Voir aussi

Tâches

Comment : faire la distinction entre deux éléments portant le même nom
Comment : masquer une variable portant le même nom que votre variable
Comment : accéder à une variable masquée par une classe dérivée

Référence

Overrides

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

Occultation dans Visual Basic
Différences entre l'occultation et la substitution

Autres ressources

Références aux éléments déclarés

Comment : accéder à une variable masquée par une classe dérivée

Lorsque le code d'une classe dérivée accède à une variable, le compilateur résout normalement la référence vers la version accessible la plus proche, à savoir la version accessible avec le moins d'étapes de dérivation vers l'arrière de la classe accédante. Si la variable est définie dans la classe dérivée, le code accède normalement à cette définition.

Si la variable de classe dérivée occulte une variable de la classe de base, elle masque la version de classe de base. Toutefois, vous pouvez accéder à la variable de classe de base en la qualifiant avec le mot clé MyBase.

Pour accéder à une variable de classe de base masquée par une classe dérivée

- Dans une instruction d'expression ou d'assignation, faites précéder le nom de la variable du mot clé **MyBase** et d'un point (**.**).

  Le compilateur résout la référence vers la version de classe de base de la variable.

  L'exemple suivant illustre l'occultation par l'intermédiaire de l'héritage. Il effectue deux références, une qui accède à la variable occultante et une qui outrepasse l'occultation.

```
Public Class shadowBaseClass
    Public shadowString As String = "This is the base class string."
End Class
Public Class shadowDerivedClass
    Inherits shadowBaseClass
    Public Shadows shadowString As String = "This is the derived class string."
    Public Sub showStrings()
        Dim s As String = "Unqualified shadowString: " & shadowString _
            & vbCrLf & "MyBase.shadowString: " & MyBase.shadowString
        MsgBox(s)
    End Sub
End Class
```

  L'exemple précédent déclare la variable `shadowString` dans la classe de base et la masque dans la classe dérivée. La procédure `showStrings` dans la classe dérivée affiche la version occultante de la chaîne lorsque le nom `shadowString` n'est pas qualifié. Elle affiche ensuite la version occultée lorsque `shadowString` est qualifié avec le mot clé MyBase.

Programmation fiable

Pour réduire le risque de référence à une version non souhaitée d'une variable occultée, vous pouvez effectuer une qualification complète de toutes les références vers une variable occultée. L'occultation introduit plusieurs versions d'une variable avec le même nom. Lorsqu'une instruction de code fait référence au nom de la variable, la version vers laquelle le compilateur résout la référence dépend de plusieurs facteurs, tels que l'emplacement de l'instruction de code et la présence d'une chaîne qualifiante. Cela peut augmenter le risque de référence à la version incorrecte de la variable.

Voir aussi

Tâches

Comment : faire la distinction entre deux éléments portant le même nom
Comment : masquer une variable portant le même nom que votre variable
Comment : masquer une variable héritée

Référence

Shadows
Overrides

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom
Occultation dans Visual Basic
Différences entre l'occultation et la substitution

Autres ressources

Références aux éléments déclarés

Types de données en Visual Basic

Le *type de données* détermine la nature des données que peut contenir un élément de programmation et la manière dont il enregistre ces données. Les types de données s'appliquent à toutes les valeurs susceptibles d'être enregistrées dans la mémoire d'un ordinateur ou d'intervenir dans l'évaluation d'une expression. Chaque variable, littéral, constante, énumération, propriété, paramètre de procédure, argument de procédure et valeur de retour de procédure possède un type de données.

Types de données déclarés

Sauf si vous utilisez la programmation sans type, vous devez déclarer les types de données pour tous les éléments de programmation. (Pour plus d'informations, consultez Programmation sans type dans Visual Basic).

Un élément de programmation est défini dans une instruction de déclaration et son type de données est spécifié à l'aide de la clause **As**. Le tableau suivant présente les instructions que vous utilisez pour déclarer divers éléments.

Élément de programmation	Déclaration de type de données

Variable	Dans Dim, instruction (Visual Basic)
	`Dim amount As Double`
	`Static yourName As String`
	`Public billsPaid As Decimal = 0`

Literal	Avec un caractère de type de littéral ; consultez « Caractères de type de littéral » dans Caractères de type `Dim searchChar As Char = "."C`
Constante	Dans Const, instruction (Visual Basic) `Const modulus As Single = 4.17825F`
Énumération	Dans Enum, instruction (Visual Basic) `Public Enum colors`
Propriété	Dans une instruction Property `Property region() As String`
Paramètre de procédure	Dans une instruction Sub (Visual Basic), Function (Visual Basic), ou Operator `Sub addSale(ByVal amount As Double)`
Argument de procédure	Dans le code appelant ; chaque argument est un élément de programmation qui a déjà été déclaré, ou une expression qui contient des éléments déclarés `subString = Left( inputString , 5 )`
Valeur de retour de procédure	Dans une instruction Function (Visual Basic) ou Operator `Function convert(ByVal b As Byte) As String`

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)

Concepts

Caractères de type

Types de données composites

Types génériques en Visual Basic

Programmation sans type dans Visual Basic

Autres ressources

Types de données élémentaires
Implémentation des types de données
Conversions de type en Visual Basic
Structures constituées à partir de vos propres types de données

Caractères de type

En plus de spécifier un type de données dans une instruction de déclaration, vous pouvez forcer le type de données de certains éléments de programmation à l'aide d'un *caractère de type*. Le caractère de type doit immédiatement suivre l'élément (aucun autre caractère ne doit venir s'interposer).

Le caractère de type ne fait pas partie du nom de l'élément. Un élément défini avec un caractère de type peut être référencé sans celui-ci.

Caractères de type d'identificateur

Visual Basic met à votre disposition des *caractères de type d'identificateur* que vous pouvez utiliser dans une déclaration pour spécifier le type de données d'une variable ou d'une constante. Le tableau suivant répertorie les caractères de type d'identificateur disponibles et donne des exemples de leur emploi.

Caractère de type d'identificateur	Type de données	Exemple
%	**Integer**	`Dim L%`
&	**Long**	`Dim M&`
@	**Decimal**	`Const W@ = 37.5`
!	**Single**	`Dim Q!`
#	**Double**	`Dim X#`
$	**String**	`Dim V$ = "Secret"`

Il n'existe aucun caractère de type d'identificateur pour les types de données **Boolean**, **Byte**, **Char**, **Date**, **Object**, **SByte**, **Short**, **UInteger**, **ULong** ou **UShort**, ni pour les types de données composites, tels que les tableaux ou les structures.

Dans certains cas, vous pouvez ajouter le caractère **$** à une fonction Visual Basic, par exemple **Left$** au lieu de **Left**, afin d'obtenir une valeur retournée de type **String**.

Dans tous les cas, le caractère de type d'identificateur doit immédiatement suivre le nom de l'identificateur.

Caractères de type de littéral

Un *littéral* est une représentation textuelle d'une valeur particulière d'un type de données.

Types de littéral par défaut

La forme d'un littéral, telle qu'elle se présente dans votre code, détermine normalement son type de données. Le tableau ci-dessous répertorie ces types par défaut.

Forme textuelle de littéral	Type de données par défaut	Exemple
Numérique, aucune partie fractionnaire	**Integer**	`2147483647`
Numérique, aucune partie fractionnaire, trop grand pour **Integer**	**Long**	`2147483648`
Numérique, partie fractionnaire	**Double**	`1.2`
Placé entre guillemets doubles	**String**	`"A"`
Placé entre signes dièse	**Date**	`#5/17/1993 9:32 AM#`

Types de littéral forcés

Visual Basic met à votre disposition un ensemble de *caractères de type de littéral* que vous pouvez utiliser pour forcer un littéral à prendre un type de données autre que celui qui est indiqué par sa forme. Pour ce faire, ajoutez le caractère souhaité

après le littéral. Le tableau suivant répertorie les caractères de type de littéral disponibles et donne des exemples de leur emploi.

Caractère de type de littéral	Type de données	Exemple
S	**Short**	`I = 347S`
I	**Integer**	`J = 347I`
L	**Long**	`K = 347L`
D	**Decimal**	`X = 347D`
F	**Single**	`Y = 347F`
R	**Double**	`Z = 347R`
US	**UShort**	`L = 347US`
UI	**UInteger**	`M = 347UI`
UL	**ULong**	`N = 347UL`
C	**Char**	`Q = "."C`

Il n'existe aucun caractère de type de littéral pour les types de données **Boolean**, **Byte**, **Date**, **Object**, **SByte** ou **String**, ni pour les types de données composites, tels que les tableaux ou les structures.

Les littéraux peuvent également utiliser les caractères de type d'identificateur (**%**, **&**, **@**, **!**, **#**, **$**), tout comme le peuvent les variables, les constantes et les expressions. Toutefois, les caractères de type de littéral (**S**, **I**, **L**, **D**, **F**, **R**, **C**) peuvent être utilisés uniquement avec des littéraux.

Dans tous les cas, le caractère de type de littéral doit immédiatement suivre la valeur littérale.

Littéraux hexadécimaux et octaux

Le compilateur interprète normalement un littéral entier comme représenté avec le système de numération décimale (base 10). Vous pouvez forcer un littéral entier à être hexadécimal (base 16) avec le préfixe **&H** et vous pouvez le forcer à être octal (base 8) avec le préfixe **&O**. Les chiffres qui suivent le préfixe doivent être adaptés au système de numération. Le tableau suivant illustre ce système de représentation.

Base numérique	Préfixe	Valeurs de chiffres valables	Exemple
Hexadécimale (base 16)	**&H**	0-9 et A-F	`&HFFFF`
Octale (base 10)	**&O**	0-7	`&O77`

Vous pouvez faire suivre un littéral préfixé d'un caractère de type de littéral. L'exemple suivant fournit une illustration.

```
Dim counter As Short = &H8000S
Dim flags As UShort = &H8000US
```

Dans l'exemple précédent, `counter` a la valeur décimale -32 768 et `flags` a la valeur décimale +32 768.

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)

Concepts

Types de données en Visual Basic

Programmation sans type dans Visual Basic
Déclaration de variable dans Visual Basic

Autres ressources

Types de données élémentaires
Implémentation des types de données
Conversions de type en Visual Basic

Types de données élémentaires

Visual Basic fournit un jeu de types de données prédéfinis que vous pouvez utiliser pour une grande partie de vos éléments de programmation. Cette section décrit ces types et explique comment les utiliser.

Dans cette section

Types de données numériques

Présente les types numériques intégraux et non intégraux.

Types de données caractère

Présente les types **Char** et **String**.

Types de données divers

Présente les types **Boolean**, **Date** et **Object**.

Rubriques connexes

Types de données en Visual Basic

Introduit les types de données Visual Basic et décrit comment les utiliser.

Liste des types de données (Visual Basic)

Propose une vue d'ensemble des types de données élémentaires disponibles dans Visual Basic.

Types de données numériques

Visual Basic met à votre disposition plusieurs *types de données numériques* permettant de gérer les nombres dans diverses représentations. Les types *intégraux* ne représentent que les nombres entiers (positifs, négatifs et zéro), tandis que les types *non intégraux* représentent les nombres composés d'une partie entière et d'une partie fractionnaire.

Pour examiner un tableau présentant une comparaison côte à côte des types de données de Visual Basic, consultez Liste des types de données (Visual Basic).

Types numériques intégraux

Les *types de données intégraux* sont ceux qui ne représentent que des nombres sans partie fractionnaire.

Les types de données intégraux *signés* sont SByte, type de données (Visual Basic) (8 bits), Short, type de données (Visual Basic) (16 bits), Integer, type de données (Visual Basic) (32 bits) et Long, type de données (Visual Basic) (64 bits). Si une variable enregistre toujours des nombres entiers plutôt que des nombres fractionnaires, déclarez-la avec l'un de ces types.

Les types intégraux *non signés* sont les types de données Byte (Visual Basic) (8 bits), UShort (Visual Basic) (16 bits), UInteger (32 bits) et ULong (Visual Basic) (64 bits). Si une variable contient des données binaires ou des données de nature inconnue, déclarez-la avec l'un de ces types.

Performance

Les opérations arithmétiques sont plus rapides avec les types intégraux qu'avec les autres types de données. Elles sont plus rapides avec les types **Integer** et **UInteger** dans Visual Basic.

Grands entiers

Si vous devez placer un entier plus grand que le type de données **Integer** ne peut contenir, vous pouvez utiliser à la place le type de données **Long**. Les variables **Long** peuvent contenir des nombres compris entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807. Les opérations avec **Long** sont légèrement plus lentes qu'avec **Integer**.

Si vous avez besoin de valeurs encore plus grandes, vous pouvez utiliser le Decimal, type de données (Visual Basic). Vous pouvez placer des nombres compris entre -79 228 162 514 264 337 593 543 950 335 et 79 228 162 514 264 337 593 543 950 335 dans une variable **Decimal** si vous n'utilisez pas de décimales. Toutefois, les opérations avec les nombres **Decimal** sont considérablement plus lentes qu'avec tout autre type de données numériques.

Petits entiers

Si vous n'avez pas besoin de la plage complète du type de données **Integer**, vous pouvez utiliser le type de données **Short**, qui peut contenir des entiers compris entre -32 768 et 32 767. Pour la plus petite plage d'entiers, le type de données **SByte** contient des entiers compris entre -128 et 127. Si vous avez un grand nombre de variables qui contiennent de petits entiers, le Common Langage Runtime peut quelquefois stocker plus efficacement vos variables **Short** et **SByte** et réduire la consommation de mémoire. Toutefois, les opérations avec **Short** et **SByte** sont un peu plus lentes qu'avec **Integer**.

Entiers non signés

Si vous savez que votre variable ne contiendra jamais de nombre négatif, vous pouvez utiliser les *types non signés* **Byte**, **UShort**, **UInteger** et **ULong**. Chacun de ces types de données peut contenir un entier positif deux fois plus grand que son type signé correspondant (**SByte**, **Short**, **Integer** et **Long**). En termes de performances, chaque type non signé est exactement aussi efficace que son type signé correspondant. En particulier, **UInteger** partage avec **Integer** la distinction d'être le plus efficace de tous les types de données numériques élémentaires.

Types numériques non intégraux

Les *types de données non intégraux* représentent des nombres composés d'une partie entière et d'une partie fractionnaire.

Les types de données numériques non intégraux sont **Decimal** (128 bits virgule fixe), Single, type de données (Visual Basic) (32 bits virgule flottante) et Double, type de données (Visual Basic) (64 bits virgule flottante). Tous ces types sont signés. Si une variable peut contenir une fraction, déclarez-la avec l'un de ces types.

Decimal n'est pas un type de données à virgule flottante. Les nombres **Decimal** ont une valeur d'entier binaire et un facteur d'échelle d'entier qui spécifie quelle partie de la valeur est une fraction décimale.

Les nombres à virgule flottante (**Single** et **Double**) présentent des plages plus larges que les nombres de type **Decimal**, mais ils peuvent être sujets aux erreurs d'arrondi. Les types à virgule flottante prennent en charge un nombre de chiffres significatifs inférieur à celui du type **Decimal**, mais ils peuvent représenter des valeurs d'amplitude supérieure.

Les valeurs de nombres non intégraux peuvent être exprimées comme *mmmEeee*, où *mmm* est la *mantisse* (les bits significatifs) et *eee* est l'*exposant* (une puissance de 10). Les valeurs positives les plus élevées des types non intégraux sont 7,9228162514264337593543950335E+28 pour **Decimal**, 3,4028235E+38 pour **Single** et 1,79769313486231570E+308 pour **Double**.

Performance

Double est le plus efficace des types de données fractionnaires, car les processeurs des plates-formes actuelles exécutent les opérations en virgule flottante en double précision. Toutefois, les opérations avec **Double** ne sont pas aussi rapides qu'avec les types intégraux, tels que **Integer**.

Petites amplitudes

Pour les nombres avec la plus petite amplitude possible (le plus proche de 0), les variables **Double** peuvent contenir des nombres aussi petits que -4,94065645841246544E-324 pour les valeurs négatives et 4,94065645841246544E-324 pour les valeurs positives.

Petits nombres fractionnaires

Si vous n'avez pas besoin de la plage complète du type de données **Double**, vous pouvez utiliser le type de données **Single**, qui peut contenir des nombres à virgule flottante compris entre -3,4028235E+38 et 3,4028235E+38. Les plus petites amplitudes pour les variables **Single** sont -1,401298E-45 pour les valeurs négatives et 1,401298E-45 pour les valeurs positives. Si vous avez un grand nombre de variables qui contiennent de petits nombres à virgule flottante, le Common Language Runtime peut quelquefois stocker plus efficacement vos variables **Single** et réduire la consommation de mémoire.

Voir aussi

Tâches

Dépannage des types de données
Comment : stocker des entiers dans une variable
Comment : stocker des fractions dans une variable
Comment : stocker le plus grand nombre possible dans une variable
Comment : optimiser le stockage d'entiers positifs avec les types non signés
Comment : appeler une fonction Windows qui possède des types non signés
Comment : stocker les chiffres les plus significatifs dans une variable
Comment : stocker des valeurs monétaires dans une variable

Concepts

Types de données caractère
Types de données divers

Autres ressources

Types de données élémentaires

Comment : stocker des entiers dans une variable

Les *entiers* sont des nombres entiers, y compris les nombres positifs, négatifs et zéro. Une variable contient des entiers si vous la déclarez comme étant de type **Integer**. Le Integer, type de données (Visual Basic) peut contenir des entiers compris entre -2 147 483 648 et 2 147 483 647.

Vous pouvez utiliser également le Long, type de données (Visual Basic), le Short, type de données (Visual Basic) ou le SByte, type de données (Visual Basic) pour spécifier des entiers. Pour plus d'informations, consultez Types de données numériques.

Pour stocker des entiers dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Integer**.

```
Dim students As Integer
```

Voir aussi

Référence

Liste des types de données (Visual Basic)

Concepts

Types de données en Visual Basic

Caractères de type

Autres ressources

Types de données élémentaires

Comment : stocker le plus grand nombre possible dans une variable

Une variable contient les plus grands nombres possibles avec précision si vous la déclarez comme un type de données **Decimal**. La plus grande capacité d'entier suivante est le type de données **ULong**. Si vous n'avez pas besoin de la précision de types de données intégraux, vous pouvez utiliser des types à virgule flottante pour une amplitude encore supérieure.

Deux significations différentes pour « le plus grand »

La plus grande valeur précise. Si vous devez stocker de grands entiers avec une précision totale jusqu'au chiffre des unités, vous pouvez utiliser Decimal, type de données (Visual Basic). Le type **Decimal** peut stocker des entiers compris entre -79 228 162 514 264 337 593 543 950 335 et 79 228 162 514 264 337 593 543 950 335 (*7.9...E+28*).

La plus grande amplitude. Les types fractionnaires **Single** et **Double** peuvent stocker des nombres de grande amplitude, mais sans précision exacte. Single, type de données (Visual Basic) fournit une précision de 8 chiffres et Double, type de données (Visual Basic) une précision de 18 chiffres.

Pour stocker les plus grands entiers possibles dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As** spécifiant le mot clé **Decimal**.

```
Dim atomsInTheUniverse As Decimal
```

Types de données efficaces

Le type **Decimal** offre les performances les plus lentes de tous les types de données numériques élémentaires. Si vos entiers n'atteignent pas des valeurs de cette taille et sont toujours positifs ou nuls, envisagez le type **ULong**.

Une variable ayant le ULong, type de données (Visual Basic) peut stocker des entiers compris entre 0 et 18 446 744 073 709 551 615 (*1.8...E+19*). Les opérations avec les nombres **ULong** sont beaucoup plus rapides qu'avec **Decimal**, même si elles ne sont pas aussi efficaces qu'avec **UInteger**.

Pour stocker de grands entiers non négatifs dans une variable avec des performances correctes

1. Déclarez la variable à l'aide d'une instruction **Dim**.
2. Faites suivre le nom de variable d'une clause **As** spécifiant le mot clé **ULong**.

```
Dim atomsInTheEarth As ULong
```

Voir aussi

Référence

Liste des types de données (Visual Basic)
Integer, type de données (Visual Basic)
Long, type de données (Visual Basic)
Decimal, type de données (Visual Basic)
UInteger, type de données
ULong, type de données (Visual Basic)

Concepts

Caractères de type

Autres ressources

Types de données élémentaires

Comment : optimiser le stockage d'entiers positifs avec les types non signés

Si vous avez une variable qui contient uniquement des valeurs positives (ou 0) et que ces valeurs ne dépassent jamais 4 294 967 295, vous pouvez déclarer la variable en tant que **UInteger** plutôt que **Long**.

L'avantage de **UInteger** est que les types entiers de 32 bits **Integer** et **UInteger** sont les types de données les plus efficaces sur les plates-formes 32 bits et qu'ils fournissent les performances optimales pour votre application.

Vous pouvez utiliser une variable **Integer** si vos valeurs positives ne dépassent jamais 2 147 483 647.

Pour déclarer un entier avec uniquement des valeurs positives

- Déclarez la variable **As UInteger**. L'exemple suivant illustre ce comportement :

```
Public Function memoryRequired(ByVal m As UInteger) As UInteger
    Static r As UInteger = 0
    Try
        r += m
    Catch eo As System.OverflowException
        r = 0
    Catch ex As System.Exception
        MsgBox("Incrementing required memory causes """ & ex.Message & """")
    End Try
    Return r
End Function
```

Vous pouvez tester la fonction `memoryRequired` à l'aide du code suivant :

```
Public Sub consumeMemoryRequired()
    Dim m1 As UInteger = UInteger.MaxValue - 100
    Dim m2 As UInteger = 100
    MsgBox("Max = " & CStr(UInteger.MaxValue) & vbCrLf & _
        CStr(m1) & " -> " & CStr(memoryRequired(m1)) & vbCrLf & _
        "+ " & CStr(m2) & " -> " & CStr(memoryRequired(m2)) _
        & vbCrLf & "+ 1 -> " & CStr(memoryRequired(1)))
End Sub
```

Attention

Le type de données **UInteger** ne faisant pas partie de Spécification CLS (Common Language Specification), le code conforme CLS ne peut pas consommer un composant qui l'utilise.

Voir aussi

Tâches

Comment : appeler une fonction Windows qui possède des types non signés

Référence

Liste des types de données (Visual Basic)

Integer, type de données (Visual Basic)

UInteger, type de données

Comment : appeler une fonction Windows qui possède des types non signés

Si vous consommez une classe, un module ou une structure qui a des membres de types d'entiers non signés, vous pouvez accéder à ces membres avec Visual Basic.

Pour appeler une fonction Windows qui prend un type non signé

1. Utilisez une Declare, instruction pour indiquer à Visual Basic quelle bibliothèque contient la fonction, quel est son nom dans cette bibliothèque, quelle est sa séquence d'appel et comment convertir des chaînes lors de son appel.
2. Dans l'instruction **Declare**, utilisez **UInteger**, **ULong**, **UShort** ou **Byte** selon le cas pour chaque paramètre avec un type non signé.
3. Pour déterminer les noms et les valeurs des constantes de la fonction Windows que vous appelez, consultez la documentation relative à cette fonction. Une grande partie de ces noms et valeurs sont définis dans le fichier WinUser.h.
4. Déclarez les constantes nécessaires dans votre code. De nombreuses constantes Windows sont des valeurs non signées de 32 bits, que vous devez déclarer **As UInteger**.
5. Appelez la fonction de manière normale. L'exemple suivant appelle la fonction Windows **MessageBox**, qui prend un argument d'entier non signé.

```
Public Class windowsMessage
    Private Declare Auto Function mb Lib "user32.dll" Alias "MessageBox" _
        (ByVal hWnd As Integer, _
        ByVal lpText As String, _
        ByVal lpCaption As String, _
        ByVal uType As UInteger) As Integer
    Private Const MB_OK As UInteger = 0
    Private Const MB_ICONEXCLAMATION As UInteger = &H30
    Private Const IDOK As UInteger = 1
    Private Const IDCLOSE As UInteger = 8
    Private Const c As UInteger = MB_OK Or MB_ICONEXCLAMATION
    Public Function messageThroughWindows() As String
        Dim r As Integer = mb(0, "Click OK if you see this!", _
            "Windows API call", c)
        Dim s As String = "Windows API MessageBox returned " _
            & CStr(r)& vbCrLf & "(IDOK = " & CStr(IDOK) _
            & ", IDCLOSE = " & CStr(IDCLOSE) & ")"
        Return s
    End Function
End Class
```

Vous pouvez tester la fonction `messageThroughWindows` à l'aide du code suivant.

```
Public Sub consumeWindowsMessage()
    Dim w As New windowsMessage
    w.messageThroughWindows()
End Sub
```

Attention

Les types de données **UInteger**, **ULong**, **UShort** et **SByte** ne faisant pas partie de la Spécification CLS (Common Language Specification) (CLS), le code conforme CLS ne peut pas consommer un composant qui les utilise.

Remarque de sécurité

Le fait d'appeler du code non managé, tel que l'interface de programmation d'application Windows (API), expose votre code à des problèmes de sécurité potentiels.

Remarque de sécurité

L'appel de l'API Windows nécessite une autorisation de code non managée, qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Voir aussi

Tâches

Comment : optimiser le stockage d'entiers positifs avec les types non signés

Procédure pas à pas : appel des API Windows

Référence

Liste des types de données (Visual Basic)

Integer, type de données (Visual Basic)

UInteger, type de données

Declare, instruction

Comment : stocker des fractions dans une variable

Une variable stocke des nombres à fraction si vous la déclarez comme un type **Double**. Le Double, type de données (Visual Basic) peut stocker des nombres à virgule flottante de -1,79769313486231570E+308 à 1,79769313486231570E+308.

Vous pouvez également utiliser Single, type de données (Visual Basic) ou Decimal, type de données (Visual Basic) pour stocker des fractions. Pour plus d'informations, consultez Types de données numériques.

Pour stocker des fractions dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Double**.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Concepts

Caractères de type

Types de données numériques

Autres ressources

Types de données élémentaires

Comment : stocker les chiffres les plus significatifs dans une variable

Une variable stocke des nombres à fraction si vous la déclarez comme un type de données **Decimal**.

Une variable du Decimal, type de données (Visual Basic) peut stocker des nombres à 29 chiffres significatifs ainsi qu'un maximum de 28 décimales.

Performances. Decimal est le moins efficace des types de données numériques. Si vous n'avez pas besoin de la précision du type de données **Decimal**, vous pouvez utiliser le type de données **Double** qui peut stocker jusqu'à 18 chiffres significatifs. Les opérations avec **Double** sont plus rapides qu'avec **Decimal**, et les variables **Double** prennent moins d'espace en mémoire. Pour plus d'informations sur **Double**, consultez Comment : stocker des fractions dans une variable.

Vous devez peser l'importance de la précision par rapport aux performances avant de choisir un type de données.

Plus grande amplitude. S'il est nécessaire que votre variable stocke des nombres d'une amplitude supérieure à ce que le type de données **Decimal** peut stocker, vous pouvez utiliser le type de données **Double** ou **Single**, chacun possédant une plus grande plage. Si les variables **Decimal** peuvent stocker des valeurs allant jusqu'à 7,9228162514264337593543950335E+28, les variables **Single** peuvent stocker jusqu'à 3,4028235E+38, et les variables **Double** jusqu'à 1,79769313486231570E+308.

Pour stocker les chiffres les plus significatifs dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Decimal**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Decimal, type de données (Visual Basic)
Double, type de données (Visual Basic)

Concepts

Caractères de type

Autres ressources

Types de données élémentaires

Comment : stocker des valeurs monétaires dans une variable

Les valeurs monétaires doivent être stockées dans des variables de type de données **Decimal**.

Une variable ayant le Decimal, type de données (Visual Basic) peut contenir des nombres avec 29 bits significatifs, y compris jusqu'à 28 décimales. Pour les valeurs monétaires, il suffit généralement de 2 ou 3 décimales. Toutefois, pour les résultats intermédiaires pendant une transaction (telle que la multiplication par un taux d'intérêt), vous souhaiterez peut-être utiliser un plus grand nombre de décimales afin de conserver un degré de précision adéquat.

L'utilisation des variables **Decimal** pour les valeurs monétaires présente un avantage au niveau de la précision des valeurs. Le type de données **Double** est plus rapide et requiert moins de mémoire, mais il est soumis aux erreurs d'arrondi. Le type de données **Decimal** garantit une exactitude totale à 28 décimales près, si nécessaire.

Caractères 0 de fin. Visual Basic n'a aucune représentation interne des caractères 0 de fin dans les types de données à virgule flottante, tels que **Double**. Il ne stocke pas de zéros de fin dans un littéral **Decimal**, bien qu'une variable **Decimal** préserve les zéros de fin acquis par le calcul. Pour obtenir des informations supplémentaires et un exemple, consultez Decimal, type de données (Visual Basic).

Visual Basic supprime souvent les caractères 0 de fin lors de l'affichage ou de l'impression. Par exemple, une valeur de 4,2000 est souvent restituée sous la forme 4,2. Vous pouvez contrôler davantage le format de sortie si vous appelez la méthode System.Decimal.ToString ou System.Double.ToString pour le type de données approprié.

Pour stocker des valeurs monétaires dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Decimal**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Single, type de données (Visual Basic)
Decimal, type de données (Visual Basic)
Double, type de données (Visual Basic)

Concepts

Types de données en Visual Basic
Caractères de type

Autres ressources

Types de données élémentaires

Types de données caractère

Visual Basic met à votre disposition des *types de données caractère* permettant de gérer les caractères affichables et imprimables. Si les deux types reconnaissent les caractères Unicode, **Char** n'accepte qu'un seul caractère, tandis que **String** peut contenir un nombre de caractères quelconque.

Pour examiner un tableau présentant une comparaison côte à côte des types de données de Visual Basic, consultez Liste des types de données (Visual Basic).

Type Char

Le type de données **Char** est un caractère Unicode à deux octets (16 bits) unique. Si une variable stocke toujours exactement un caractère, déclarez-la comme **Char**.

Pour plus d'informations, consultez Char, type de données (Visual Basic).

Type String

Le type de données **String** est une séquence de zéro ou de plusieurs caractères Unicode à deux octets (16 bits). Si une variable peut contenir un nombre indéfini de caractères, déclarez-la comme **String**.

Pour plus d'informations, consultez String, type de données (Visual Basic).

Voir aussi

Tâches

Dépannage des types de données
Comment : stocker des caractères dans une variable

Concepts

Types de données composites
Types génériques en Visual Basic
Programmation sans type dans Visual Basic
Caractères de type

Autres ressources

Types de données élémentaires
Implémentation des types de données
Conversions de type en Visual Basic

Comment : stocker des caractères dans une variable

Une variable stocke des caractères si vous la déclarez comme étant du type de données **Char** ou **String**.

Une variable ayant le Char, type de données (Visual Basic) peut contenir un seul caractère Unicode. Une variable ayant le String, type de données (Visual Basic) peut contenir une séquence de zéro ou plusieurs caractères Unicode.

Caractères Unicode. Chaque valeur possible dans une variable **Char** ou **String** est un *point de code*, ou code de caractère, dans le jeu de caractères Unicode. Les caractères Unicode incluent le jeu de caractères ASCII de base, plusieurs autres lettres de l'alphabet, des accents, des symboles monétaires, des fractions, des signes diacritiques ainsi que des symboles mathématiques et techniques.

Si vous n'avez jamais besoin de placer plus d'un caractère dans une variable, utilisez le type de données **Char**. Le type de données **String** requiert plus de mémoire et ralentit les performances.

Remarque
Le jeu de caractères Unicode réserve les points de code D800 à DFFF (55 296 à 55 551 décimal) pour les *paires de substitution*, qui requièrent deux valeurs de 16 bits pour représenter un seul point de code. Une variable **Char** ne peut pas contenir de paire de substitution et une variable **String** utilise deux positions pour contenir une telle paire.

Pour stocker un caractère unique dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Char**.

Pour stocker une séquence de caractères dans une variable

1. Déclarez la variable à l'aide d'une instruction **Dim**.
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **String**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Single, type de données (Visual Basic)
Char, type de données (Visual Basic)
String, type de données (Visual Basic)

Concepts

Types de données en Visual Basic
Caractères de type

Autres ressources

Types de données élémentaires

Types de données divers

Visual Basic met à votre disposition plusieurs types de données qui ne correspondent ni à des nombres ni à des caractères. Ces types de données correspondent à des données particulières, telles que des valeurs oui/non, des valeurs date/heure et des adresses d'objets.

Pour examiner un tableau présentant une comparaison côte à côte des types de données de Visual Basic, consultez Liste des types de données (Visual Basic).

Type Boolean

Le Boolean, type de données (Visual Basic) est une valeur non signée qui est interprétée comme **True** ou **False**. Sa largeur de données dépend de la plate-forme d'implémentation. Si une variable est destinée à ne contenir que des informations à deux états de type true/false, yes/no ou on/off, déclarez-la comme **Boolean**.

Type Date

Le Type de données date (Visual Basic) est une valeur 64 bits qui contient des informations d'horodatage. Chaque incrément représente 100 nanosecondes du temps écoulé depuis le 1er janvier (0 heure) de l'an 1 du calendrier grégorien. Si une variable peut contenir une valeur de date, une valeur d'heure ou les deux, déclarez-la comme **Date**.

Type Object

Le type de données Object correspond à une adresse de 32 bits pointant vers une instance d'objet dans une application (la vôtre ou une autre). Une variable **Object** peut faire référence à tout objet que votre application reconnaît ou à des données de tout type de données. Si une variable stocke un pointeur vers une instance d'une classe que vous ne connaissez pas au moment de la compilation ou si elle peut pointer vers des données de plusieurs types de données, déclarez-la comme **Object**.

Voir aussi

Tâches

Dépannage des types de données
Comment : stocker des valeurs True et False dans une variable
Comment : stocker des valeurs de date et d'heure dans une variable
Comment : stocker des données de type inconnu dans une variable

Concepts

Caractères de type
Types de données numériques
Types de données caractère

Autres ressources

Types de données élémentaires

Comment : stocker des valeurs True et False dans une variable

Une variable contient des valeurs vraies et fausses si vous la déclarez comme étant de type de données **Boolean**.

Une variable du Boolean, type de données (Visual Basic) peut contenir des valeurs logiques à deux états, tels que vrai/faux, activé/désactivé et oui/non.

Si vous devez stocker des valeurs numériques, utilisez un type de données numériques, tel que **Integer** ou **Double** plutôt que **Boolean**. Vous ne devez jamais utiliser de valeurs **Boolean** comme nombres et vous ne devez jamais compter sur des valeurs numériques équivalentes pour elles. Pour plus d'informations, consultez Dépannage des types de données.

Pour stocker des valeurs vraies et fausses dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Boolean**.

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)
Single, type de données (Visual Basic)
Boolean, type de données (Visual Basic)
Integer, type de données (Visual Basic)
Double, type de données (Visual Basic)

Concepts

Types de données en Visual Basic
Caractères de type

Autres ressources

Types de données élémentaires

Comment : stocker des valeurs de date et d'heure dans une variable

Une variable stocke des valeurs de date et d'heure si vous déclarez son type de données comme étant **Date**.

Une variable de Type de données date (Visual Basic) peut contenir des valeurs de date comprises entre le 1er janvier de l'année 0001 et le 31 décembre de l'année 9999, et des valeurs d'heure comprises entre 00:00:00 (minuit) et 23:59:59.

Le type de données **Date** comporte toujours une valeur de date et une valeur d'heure. Si vous n'assignez pas de valeur d'heure à une variable **Date**, par défaut, elle sera égale au début de la journée, c'est-à-dire minuit. Si vous n'assignez pas de valeur de date à une variable **Date**, par défaut, elle sera égale au 1er janvier 0001.

Pour stocker des dates dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Date**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Single, type de données (Visual Basic)
Type de données date (Visual Basic)

Concepts

Types de données en Visual Basic
Caractères de type

Autres ressources

Types de données élémentaires

Comment : stocker des données de type inconnu dans une variable

Une variable contient des pointeurs vers des données si vous la déclarez comme un type de données **Object**.

Une variable de type de données Object peut contenir un pointeur vers une valeur de tout type de données. Cela inclut les deux *types valeur*, tels que **Integer**, **Boolean** et des instances de structure, et des *types référence* qui sont des instances d'objets créés à partir de classes, telles que **String**, Form et des instances de tableau.

Performances. L'avantage du type de données **Object** est que vous pouvez l'utiliser pour pointer vers des données de n'importe quel type de données. L'inconvénient est que vous subissez des opérations supplémentaires qui allongent le temps d'exécution et ralentissent votre application. Si vous utilisez une variable **Object** pour des types valeur, vous subissez des conversions *boxing* et *unboxing*. Si vous l'utilisez pour des types référence, vous subissez des *liaisons tardives*.

Vous devez utiliser le type de données **Object** lorsque votre variable doit contenir différents types de données à des moments différents ou lorsque vous ne savez pas au moment de la compilation quel type ou types de données votre variable peut contenir.

Pour stocker des données de type inconnu dans une variable

1. Déclarez la variable avec Dim, instruction (Visual Basic).
2. Faites suivre le nom de variable d'une clause **As**.
3. Faites suivre le mot clé **As** du mot clé **Object**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Single, type de données (Visual Basic)
Object, type de données

Concepts

Types de données en Visual Basic
Caractères de type
Types valeur et types référence

Autres ressources

Types de données élémentaires

Types de données composites

En plus des types de données élémentaires fournis par Visual Basic, vous pouvez assembler des éléments de différents types pour créer des *types de données composites*, tels que structures, tableaux et classes. Vous pouvez créer des types de données composites à partir des types élémentaires et d'autres types composites. Par exemple, vous pouvez définir un tableau d'éléments de structure ou une structure de membres de tableau.

Types de données

Un type composite diffère du type de données de l'ensemble de ses composants. Par exemple, un tableau d'éléments **Integer** n'est pas du type de données **Integer**.

Un type de données de tableau est représenté normalement à l'aide du type d'élément, de parenthèses et de virgules si nécessaire. Par exemple, un tableau unidimensionnel d'éléments **String** est représenté comme **String()** et un tableau à deux dimensions d'éléments **Boolean** est représenté comme **Boolean(,)**.

Types structure

Il n'existe pas de type de données englobant toutes les structures. Au lieu de cela, chaque définition d'une structure représente un type de données unique, même si deux structures définissent des éléments identiques dans le même ordre. Toutefois, si vous créez plusieurs instances de la même structure, Visual Basic les considère comme étant du même type de données.

Types tableau

Il n'existe pas de type de données unique englobant tous les tableaux. Le type de données d'une instance particulière d'un tableau est déterminé par les facteurs suivants :

- le fait qu'il s'agit d'un tableau ;
- le rang (nombre de dimensions) du tableau ;
- le type des éléments du tableau.

En particulier, la longueur d'une dimension ne fait pas partie du type de données de l'instance. L'exemple suivant illustre ce comportement :

```
Dim arrayA( ) As Byte = New Byte(12) {}
Dim arrayB( ) As Byte = New Byte(100) {}
Dim arrayC( ) As Short = New Short(100) {}
Dim arrayD( , ) As Short
Dim arrayE( , ) As Short = New Short(4, 10) {}
```

Dans l'exemple précédent, les variables tableau `arrayA` et `arrayB` sont considérées comme appartenant au même type de données (**Byte()**), bien qu'elles soient initialisées avec des longueurs différentes. Les variables `arrayB` et `arrayC` ne sont pas du même type car leurs types d'éléments sont différents. Les variables `arrayC` et `arrayD` ne sont pas du même type car leurs rangs sont différents. Les variables `arrayD` et `arrayE` ont le même type (**Short(,)**) car leurs rangs et types d'éléments sont les mêmes, bien que `arrayD` ne soit pas encore initialisé.

Pour plus d'informations sur les tableaux, consultez Tableaux dans Visual Basic.

Types classe

Il n'existe pas de type de données englobant toutes les classes. Bien qu'une classe puisse hériter d'une autre classe, chacune est un type de données distinct. Les différentes instances de la même classe sont du même type de données. Si vous assignez une variable d'instance de classe à une autre, elles ont non seulement le même type de données, mais elles pointent vers la même instance de classe en mémoire.

Pour plus d'informations sur les classes, consultez Présentation des classes.

Voir aussi

Tâches

Dépannage des types de données
Comment : stocker plusieurs valeurs dans une variable

Concepts

Types de données en Visual Basic
Types génériques en Visual Basic
Programmation sans type dans Visual Basic
Les types de données sont des types .NET Framework

Autres ressources

Types de données élémentaires
Implémentation des types de données
Conversions de type en Visual Basic
Structures constituées à partir de vos propres types de données

Comment : stocker plusieurs valeurs dans une variable

Une variable contient plusieurs valeurs si vous la déclarez comme étant un *type de données composite*.

Les Types de données composites incluent des structures, des tableaux et des classes. Une variable d'un type de données composite peut contenir une combinaison de types de données élémentaires et d'autres types composites. Les structures et les classes peuvent contenir du code aussi bien que des données.

Pour stocker plusieurs valeurs dans une variable

1. Déterminez le type de données composite que vous souhaitez utiliser pour votre variable.
2. Si le type de données composite n'est pas déjà défini, définissez-le afin que votre variable puisse l'utiliser.
   - Définissez une structure à l'aide d'une instruction Structure.
   - Définissez un tableau avec une instruction Dim (Visual Basic).
   - Définissez une classe avec une instruction Class (Visual Basic).
3. Déclarez votre variable avec une instruction **Dim**.
4. Faites suivre le nom de variable d'une clause **As**.
5. Faites suivre le mot clé **As** du nom du type de données composite approprié.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Concepts

Caractères de type

Types de données composites

Les types de données sont des types .NET Framework

Autres ressources

Structures constituées à partir de vos propres types de données

Tableaux dans Visual Basic

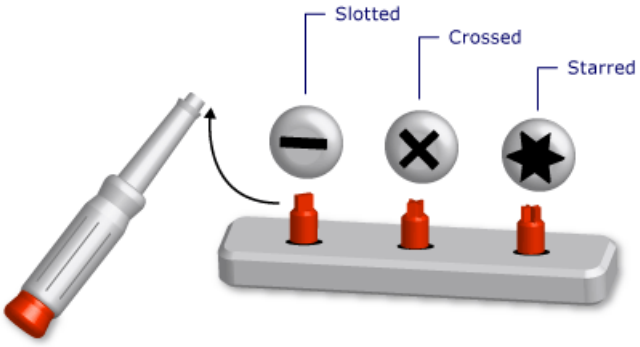
Présentation des classes

Types génériques en Visual Basic

Un *type générique* est un élément de programmation unique qui s'adapte pour exécuter les mêmes fonctionnalités pour différents types de données. Lorsque vous définissez une classe ou une procédure générique, vous ne devez pas définir une version distincte pour chaque type de données pour lequel vous souhaitez exécuter ces fonctionnalités.

On pourrait comparer cet élément de programmation à un tournevis à têtes amovibles interchangeables. Vous pouvez examiner la vis que vous devez serrer et sélectionner la tête la mieux adaptée (fendue, en croix, en étoile). Quelle que soit la tête insérée dans le manche du tournevis, vous exécutez toujours la même fonction : vous serrez une vis.

Tournevis configuré comme un outil générique



Lorsque vous définissez un type générique, vous devez le paramétrer avec un ou plusieurs types de données. Cela permet au code d'utilisation d'adapter les types de données à ses exigences. Votre code peut déclarer plusieurs éléments de programmation différents à partir de l'élément générique, agissant chacun sur un jeu différent de types de données. Mais les éléments déclarés exécutent tous la même logique, quels que soient les types de données qu'ils utilisent.

Par exemple, vous souhaitez peut-être créer et utiliser une classe de file d'attente qui utilise un type de données spécifique, par exemple **String**. Vous pouvez déclarer cette classe à partir de System.Collections.Generic.Queue, comme indiqué dans l'exemple suivant.

VB

```
Public stringQ As New System.Collections.Generic.Queue(Of String)
```

Vous pouvez maintenant utiliser `stringQ` pour travailler exclusivement avec des valeurs **String**. Étant donné que `stringQ` est spécifique à **String** au lieu d'être généralisé pour les valeurs **Object**, aucune liaison tardive ou conversion de type n'est possible. Cela permet de réduire le délai d'exécution et les erreurs d'exécution.

Pour plus d'informations sur l'utilisation d'un type générique, consultez Comment : utiliser une classe générique.

Exemple d'une classe générique

L'exemple suivant présente une définition squelette d'une classe générique.

VB

```
Public Class classHolder(Of t)
    Public Sub processNewItem(ByVal newItem As t)
        Dim tempItem As t
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Dans le précédent squelette, `t` est un *paramètre de type*, c'est-à-dire un espace réservé pour un type de données que vous fournissez lors de la déclaration de la classe. À tout autre endroit de votre code, vous pouvez déclarer diverses versions de `classHolder` en fournissant divers types de données pour `t`. L'exemple suivant illustre ces deux déclarations.

VB

```
Public integerClass As New classHolder(Of Integer)
```

```
Friend stringClass As New classHolder(Of String)
```

Les instructions précédentes déclarent des *classes construites* dans lesquelles un type spécifique remplace le paramètre de type. Ce remplacement est propagé partout dans le code de la classe construite. L'exemple suivant montre à quoi ressemblera la procédure `processNewItem` dans `integerClass`.

VB

```
Public Sub processNewItem(ByVal newItem As Integer)
    Dim tempItem As Integer
    ' Inserted code now processes an Integer item.
End Sub
```

Pour un exemple plus complet, consultez
Comment : définir une classe qui fournisse des fonctionnalités identiques pour différents types de données.

Éléments de programmation disponibles

Vous pouvez définir et utiliser des classes, des structures, des interfaces, des procédures et des délégués génériques. Notez que le .NET Framework définit plusieurs classes, structures et interfaces génériques qui représentent des éléments génériques couramment utilisés. L'espace de noms System.Collections.Generic fournit des dictionnaires, des listes, des files d'attente et des piles. Avant de définir votre propre élément générique, vérifiez s'il est déjà disponible dans **System.Collections.Generic**.

Les procédures ne sont pas des types, mais vous pouvez définir et utiliser des procédures génériques. Consultez
Procédures génériques dans Visual Basic.

Avantages des types génériques

Un type générique sert de base de déclaration de plusieurs éléments de programmation différents, chacun utilisant un type de données spécifique. Les alternatives à un type générique sont :

1. un type unique utilisant le type de données **Object** ;
2. un ensemble de versions *spécifiques au type* du type, chaque version étant individuellement codée et utilisant un type de données spécifique tel que **String**, **Integer** ou un type défini par l'utilisateur tel que `customer`.

Un type générique présente les avantages suivants par rapport à ces alternatives :

- **Sécurité de type.** Les types génériques appliquent une vérification de type au moment de la compilation. Les types basés sur **Object** acceptent n'importe quel type de données, et vous devez écrire le code pour vérifier si un type de données d'entrée est acceptable. Avec les types génériques, le compilateur peut intercepter les incompatibilités de type avant l'exécution.
- **Performances.** Les types génériques ne doivent pas effectuer de conversion *box* et *unbox* des données, car chacun d'eux est spécialisé pour un type de données. Les opérations basées sur **Object** doivent effectuer une conversion box des types de données d'entrée pour les convertir en **Object** et effectuer une conversion unbox des données destinées à la sortie. Les conversions boxing et unboxing réduisent les performances.

  Les types basés sur **Object** sont également à liaison tardive, ce qui signifie que l'accès à leurs membres requiert un code supplémentaire au moment de l'exécution. Cela réduit également les performances.
- **Consolidation du code.** Le code d'un type générique doit être défini une seule fois. Un ensemble de versions spécifiques au type d'un type doit répliquer le même code dans chaque version, la seule différence étant le type de données spécifique pour cette version. Avec les types génériques, les versions spécifiques au type sont toutes générées à partir du type générique d'origine.
- **Réutilisation du code.** Le code qui ne dépend pas d'un type de données particulier peut être réutilisé avec divers types de données, s'il est générique. Vous pouvez généralement le réutiliser même avec un type de données que vous n'avez pas prévu initialement.
- **Prise en charge IDE.** Lorsque vous utilisez un type construit déclaré à partir d'un type générique, l'environnement de développement intégré (IDE) offre une plus grande prise en charge lorsque vous développez votre code. Par exemple, IntelliSense™ peut vous indiquer les options spécifiques au type pour un argument à un constructeur ou une méthode.

- **Algorithmes génériques.** Les algorithmes abstraits qui sont indépendants du type se prêtent bien aux types génériques. Par exemple, une procédure générique qui trie des éléments à l'aide de l'interface IComparable peut être utilisée avec un type de données qui implémente **IComparable**.

Contraintes

Bien que le code d'une définition de type générique doive être aussi indépendant du type que possible, certaines fonctionnalités doivent être requises pour tout type de données fourni à votre type générique. Par exemple, si vous souhaitez comparer deux éléments à des fins de tri ou de classement, leur type de données doit implémenter l'interface **IComparable**. Vous pouvez appliquer cette exigence en ajoutant une *contrainte* au paramètre de type.

Exemple d'une contrainte

L'exemple suivant présente une définition squelette d'une classe avec une contrainte qui exige que l'argument de type implémente **IComparable**.

VB

```
Public Class itemManager(Of t As IComparable)
    ' Insert code that defines class members.
End Class
```

Si le code suivant essaie de construire une classe de `itemManager` en fournissant un type qui n'implémente pas **IComparable**, le compilateur signale une erreur.

Types de contraintes

Votre contrainte peut spécifier les éléments requis suivants selon n'importe quelle combinaison :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit être du type d'une classe ou hériter d'une classe au plus
- L'argument de type doit exposer un constructeur sans paramètre accessible au code qui crée des objets à partir de celui-ci
- L'argument de type doit être un *type référence* ou un *type valeur*

Si vous devez imposer plusieurs exigences, vous utilisez une *liste de contraintes* séparée par des virgules et mise entre accolades (**{ }**). Pour exiger un constructeur accessible, vous incluez le mot clé New (Visual Basic) dans la liste. Pour exiger un type référence, vous incluez le mot clé Class (Visual Basic) ; pour exiger un type valeur, vous incluez le mot clé Structure (Visual Basic).

Pour plus d'informations sur les contraintes, consultez Liste de types.

Exemple de plusieurs contraintes

L'exemple suivant présente une définition squelette d'une classe générique avec une liste de contraintes sur le paramètre de type. Dans le code qui crée une instance de cette classe, l'argument de type doit implémenter les interfaces **IComparable** et IDisposable, être un type référence et exposer un constructeur sans paramètre accessible.

VB

```
Public Class thisClass(Of t As {IComparable, IDisposable, Class, New})
    ' Insert code that defines class members.
End Class
```

Termes importants

Les types génériques présentent et utilisent les termes suivants :

- *Type générique*. Définition d'une classe, structure, interface, procédure ou délégué pour laquelle vous fournissez au moins un type de données lors de sa déclaration.
- *Paramètre de type*. Dans une définition de type générique, espace réservé pour un type de données fourni lors de la déclaration du type.

- *Argument de type*. Type de données spécifique qui remplace un paramètre de type lorsque vous déclarez un type construit d'un type générique.
- *Contrainte*. Condition sur un paramètre de type qui restreint l'argument de type que vous pouvez lui fournir. Une contrainte peut exiger que l'argument de type implémente une interface particulière, soit ou hérite d'une classe particulière, contienne un constructeur sans paramètre accessible ou soit un type référence ou un type valeur. Vous pouvez associer ces contraintes, mais vous pouvez spécifier une classe au plus.
- *Type construit* Classe, structure, interface, procédure ou délégué déclaré à partir d'un type générique en fournissant des arguments de type pour ses paramètres de type.

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)
Of
As
Object, type de données

Concepts

Types de données en Visual Basic
Caractères de type
Types valeur et types référence

Autres ressources

Implémentation des types de données
Conversions de type en Visual Basic

Comment : définir une classe qui fournisse des fonctionnalités identiques pour différents types de données

Vous pouvez définir une classe à partir de laquelle vous pouvez créer des objets qui fournissent des fonctionnalités identiques sur différents types de données. Pour ce faire, vous spécifiez un ou plusieurs *paramètres de type* dans la définition. La classe peut servir ensuite de modèle aux objets qui utilisent divers types de données. Une classe ainsi définie est appelée *classe générique*.

L'avantage de définir une classe générique est que vous la définissez une seule fois, et votre code peut l'utiliser pour créer plusieurs objets qui utilisent divers types de données. Cela se traduit par de meilleures performances que si vous définissez la classe avec le type **Object**.

En plus des classes, vous pouvez également définir et utiliser des structures, des interfaces, des procédures et des délégués génériques.

Pour définir une classe avec un paramètre de type

1. Définissez normalement la classe.
2. Ajoutez **(Of** *typeparameter***)** immédiatement après le nom de la classe pour spécifier un paramètre de type.
3. Si vous disposez de plusieurs paramètres de type, créez une liste avec la virgule comme séparateur à l'intérieur des parenthèses. Ne répétez pas le mot clé **Of**.
4. Si votre code exécute des opérations sur un paramètre de type autre qu'une simple assignation, faites suivre ce paramètre de type d'une clause **As** pour ajouter une ou plusieurs *contraintes*. Une contrainte garantit que le type fourni pour ce paramètre de type satisfait les exigences suivantes :

   - prise en charge d'une opération, par exemple **>**, que votre code exécute ;
   - prise en charge d'un membre, par exemple une méthode, auquel votre code accède ;
   - exposition d'un constructeur sans paramètre.

   Si vous ne spécifiez pas de contraintes, les seuls opérations et membres que votre code peut utiliser sont ceux pris en charge par letype de données Object. Pour plus d'informations, consultez Liste de types.
5. Identifiez chaque membre de classe qui doit être déclaré avec un type fourni, et déclarez-le **As** *typeparameter*. Cela s'applique au stockage interne, aux paramètres de procédure et aux valeurs de retour.
6. Vérifiez que votre code utilise uniquement des opérations et des méthodes prises en charge par n'importe quel type de données qu'il peut fournir à `itemType`.

   L'exemple suivant définit une classe qui gère une liste très simple. Cette liste est contenue dans un tableau interne `items`, et le code d'utilisation peut déclarer le type de données des éléments de la liste. Un constructeur paramétré permet au code d'utilisation de définir la limite supérieure de `items`, et le constructeur par défaut lui affecte la valeur 9 (pour un total de 10 éléments).

   VB

```
Public Class simpleList(Of itemType)
  Private items() As itemType
  Private top As Integer
  Private nextp As Integer
  Public Sub New()
    Me.New(9)
  End Sub
  Public Sub New(ByVal t As Integer)
    MyBase.New()
    items = New itemType(t) {}
    top = t
    nextp = 0
  End Sub
```

```
  Public Sub add(ByVal i As itemType)
    insert(i, nextp)
  End Sub
  Public Sub insert(ByVal i As itemType, ByVal p As Integer)
    If p > nextp OrElse p < 0 Then
      Throw New System.ArgumentOutOfRangeException("p", _
        " less than 0 or beyond next available list position")
    ElseIf nextp > top Then
      Throw New System.ArgumentException("No room to insert at ", _
        "p")
    ElseIf p < nextp Then
      For j As Integer = nextp To p + 1 Step -1
        items(j) = items(j - 1)
      Next j
    End If
    items(p) = i
    nextp += 1
  End Sub
  Public Sub remove(ByVal p As Integer)
    If p >= nextp OrElse p < 0 Then
        Throw New System.ArgumentOutOfRangeException("p", _
            " less than 0 or beyond last list item")
    ElseIf nextp = 0 Then
        Throw New System.ArgumentException("List empty; cannot remove ", _
            "p")
    ElseIf p < nextp - 1 Then
        For j As Integer = p To nextp - 2
            items(j) = items(j + 1)
        Next j
    End If
    nextp -= 1
  End Sub
  Public ReadOnly Property listLength() As Integer
    Get
      Return nextp
    End Get
  End Property
  Public ReadOnly Property listItem(ByVal p As Integer) As itemType
    Get
      If p >= nextp OrElse p < 0 Then
        Throw New System.ArgumentOutOfRangeException("p", _
          " less than 0 or beyond last list item")
        End If
      Return items(p)
    End Get
  End Property
End Class
```

Vous pouvez déclarer une classe de `simpleList` pour contenir une liste de valeurs **Integer**, une autre classe pour contenir une liste de valeurs **String** et une autre pour contenir des valeurs **Date**. À l'exception du type de données des membres de la liste, les objets créés à partir de toutes ces classes ont un comportement identique.

L'argument de type que le code d'utilisation fournit à `itemType` peut être un type intrinsèque tel que **Boolean** ou **Double**, une structure, une énumération ou un type de classe, y compris l'un de ceux définis par votre application.

Vous pouvez tester la classe `simpleList` à l'aide du code suivant.

VB

```
Public Sub useSimpleList()
  Dim iList As New simpleList(Of Integer)(2)
  Dim sList As New simpleList(Of String)(3)
  Dim dList As New simpleList(Of Date)(2)
  iList.add(10)
  iList.add(20)
  iList.add(30)
  sList.add("First")
  sList.add("extra")
  sList.add("Second")
  sList.add("Third")
  sList.remove(1)
  dList.add(#1/1/2003#)
  dList.add(#3/3/2003#)
  dList.insert(#2/2/2003#, 1)
  Dim s As String = _
    "Simple list of 3 Integer items (reported length " _
    & CStr(iList.listLength) & "):" _
    & vbCrLf & CStr(iList.listItem(0)) _
    & vbCrLf & CStr(iList.listItem(1)) _
    & vbCrLf & CStr(iList.listItem(2)) _
    & vbCrLf _
    & "Simple list of 4 - 1 String items (reported length " _
    & CStr(sList.listLength) & "):" _
    & vbCrLf & CStr(sList.listItem(0)) _
    & vbCrLf & CStr(sList.listItem(1)) _
    & vbCrLf & CStr(sList.listItem(2)) _
    & vbCrLf _
    & "Simple list of 2 + 1 Date items (reported length " _
    & CStr(dList.listLength) & "):" _
    & vbCrLf & CStr(dList.listItem(0)) _
    & vbCrLf & CStr(dList.listItem(1)) _
    & vbCrLf & CStr(dList.listItem(2))
  MsgBox(s)
End Sub
```

Voir aussi

Tâches

Comment : utiliser une classe générique

Référence

Of

Liste de types

Object, type de données

Concepts

Types de données en Visual Basic

Types génériques en Visual Basic

Spécification CLS (Common Language Specification)

Comment : utiliser une classe générique

Une classe qui accepte des *paramètres de type* est appelée une *classe générique*. Si vous utilisez une classe générique, vous pouvez générer une *classe construite* à partir de celle-ci en fournissant un *argument de type* pour chacun de ces paramètres. Vous pouvez déclarer ensuite une variable du type de la classe construite, et vous pouvez créer une instance de la classe construite et l'assigner à cette variable.

En plus des classes, vous pouvez également définir et utiliser des structures, des interfaces, des procédures et des délégués génériques.

La procédure suivante accepte une classe générique définie dans le .NET Framework et crée une instance de celle-ci.

Pour utiliser une classe qui accepte un paramètre de type

1. Au début de votre fichier source, incluez une instruction Imports pour importer l'espace de noms System.Collections.Generic. Cela vous permet de faire référence à la classe System.Collections.Generic.Queue sans devoir la qualifier pleinement pour la différencier des autres classes de file d'attente telles que System.Collections.Queue.

2. Créez normalement l'objet, mais ajoutez **(Of** *type***)** immédiatement après le nom de la classe.

   L'exemple suivant utilise la même classe (**System.Collections.Generic.Queue**) pour créer deux objets Queue qui contiennent des éléments de types de données différents. Il ajoute des éléments à la fin de chaque file d'attente puis supprime et affiche les éléments présents à l'avant de chaque file d'attente.

   VB

```
Public Sub usequeue()
  Dim queueDouble As New System.Collections.Generic.Queue(Of Double)
  Dim queueString As New System.Collections.Generic.Queue(Of String)
  queueDouble.Enqueue(1.1)
  queueDouble.Enqueue(2.2)
  queueDouble.Enqueue(3.3)
  queueDouble.Enqueue(4.4)
  queueString.Enqueue("First string of three")
  queueString.Enqueue("Second string of three")
  queueString.Enqueue("Third string of three")
  Dim s As String = "Queue of Double items (reported length " _
    & CStr(queueDouble.Count) & "):"
  For i As Integer = 1 To queueDouble.Count
    s &= vbCrLf & CStr(queueDouble.Dequeue())
  Next i
  s &= vbCrLf & "Queue of String items (reported length " _
    & CStr(queueString.Count) & "):"
  For i As Integer = 1 To queueString.Count
    s &= vbCrLf & queueString.Dequeue()
  Next i
  MsgBox(s)
End Sub
```

Voir aussi

Tâches

Comment : définir une classe qui fournisse des fonctionnalités identiques pour différents types de données

Référence

Of

Imports, instruction

Concepts

Types de données en Visual Basic

Types génériques en Visual Basic

Spécification CLS (Common Language Specification)

Procédures génériques dans Visual Basic

Une *procédure générique*, appelée également *méthode générique*, est une procédure définie avec au moins un paramètre de type. Cela permet au code appelant d'adapter les types de données à ses besoins à chaque fois qu'il appelle la procédure.

Une procédure n'est pas générique simplement du fait qu'elle est définie au sein d'une classe ou d'une structure générique. Pour être générique, la procédure doit prendre au moins un paramètre de type, en plus de tous les paramètres normaux qu'il peut prendre. Une classe ou une structure générique peut contenir des procédures non génériques. Une classe, une structure ou un module non générique peut contenir des procédures génériques.

Une procédure générique peut utiliser ses paramètres de type dans sa liste de paramètres normale, dans son type de retour le cas échéant et dans son code de procédure.

Inférence de type

Vous pouvez appeler une procédure générique sans fournir du tout d'arguments de type. Si vous l'appelez de cette façon, le compilateur essaie de déterminer les types de données appropriés à passer aux arguments de type de la procédure. C'est ce qu'on appelle l'*inférence de type*. Le code suivant affiche un appel dans lequel le compilateur infère qu'il doit passer le type **String** au paramètre de type `t`.

VB

```
Public Sub testSub(Of t)(ByVal arg As t)
End Sub
Public Sub callTestSub()
    testSub("Use this string")
End Sub
```

Si le compilateur ne peut pas inférer les arguments de type à partir du contexte de votre appel, il signale une erreur. Une cause possible d'une telle erreur est une incompatibilité de rang de tableau. Par exemple, supposons que vous définissez un paramètre normal comme un tableau de paramètre de type. Si vous appelez la procédure générique fournissant un tableau d'un rang différent (nombre de dimensions), l'incompatibilité entraîne l'échec de l'inférence du type. Le code suivant affiche un appel dans lequel un tableau à deux dimensions est passé à une procédure qui attend un tableau unidimensionnel.

```
Public Sub demoSub(Of t)(ByVal arg() As t)
End Sub
Public Sub callDemoSub()
Dim twoDimensions(,) As Integer
demoSub(twoDimensions)
End Sub
```

Vous pouvez appeler l'inférence de type uniquement en omettant tous les arguments de type. Si vous fournissez un argument de type, vous devez les fournir tous.

L'inférence de type est prise en charge uniquement pour les procédures génériques. Vous ne pouvez pas appeler l'inférence de type sur les classes, les structures, les interfaces ou les délégués génériques.

Exemple

L'exemple suivant définit une procédure **Function** générique destinée à rechercher un élément particulier dans un tableau. Il définit un paramètre de type et l'utilise pour construire les deux paramètres dans la liste de paramètres.

VB

```
Public Function findElement(Of T As IComparable) _
    (ByVal searchArray As T(), ByVal searchValue As T) As Integer
    If searchArray.GetLength(0) > 0 Then
        For i As Integer = 0 To searchArray.GetUpperBound(0)
            If searchArray(i).CompareTo(searchValue) = 0 Then Return i
        Next i
    End If
    Return -1
End Function
```

L'exemple précédent nécessite la capacité à comparer `searchValue` par rapport à chaque élément de `searchArray`. Pour garantir cette capacité, il contraint le paramètre de type `T` à implémenter l'interface IComparable. Le code utilise la méthode CompareTo au lieu de l'opérateur **=**, parce qu'il n'y a aucune garantie qu'un argument de type fourni pour `T` prenne en charge l'opérateur **=**.

Vous pouvez tester la procédure `findElement` à l'aide du code suivant.

VB

```
Public Sub tryFindElement()
    Dim stringArray() As String = {"abc", "def", "xyz"}
    Dim stringSearch As String = "abc"
    Dim integerArray() As Integer = {7, 8, 9}
    Dim integerSearch As Integer = 8
    Dim dateArray() As Date = {#4/17/1969#, #9/20/1998#, #5/31/2004#}
    Dim dateSearch As Date = Microsoft.VisualBasic.DateAndTime.Today
    MsgBox(CStr(findElement(Of String)(stringArray, stringSearch)))
    MsgBox(CStr(findElement(Of Integer)(integerArray, integerSearch)))
    MsgBox(CStr(findElement(Of Date)(dateArray, dateSearch)))
End Sub
```

Les appels précédents à **MsgBox** affichent "0", "1" et "-1" respectivement.

Voir aussi

Tâches

Comment : définir une classe qui fournisse des fonctionnalités identiques pour différents types de données

Comment : utiliser une classe générique

Référence

Liste de types

Liste de paramètres

Concepts

Types génériques en Visual Basic

Procédures dans Visual Basic

Paramètres et arguments d'une procédure

Types valeur dont la valeur n'est pas définie

Vous utilisez parfois un type valeur dont la valeur n'est pas définie dans certains cas. Par exemple, un champ d'une base de données devra peut-être faire la distinction entre l'assignation d'une valeur explicite et l'absence d'assignation de valeur. Les types valeur peuvent être étendus pour accepter leurs valeurs normales ou une valeur null. Cette extension est appelée *type nullable*.

Chaque type nullable est construit à partir de la structure Nullable générique. Prenons une base de données qui assure le suivi des activités professionnelles. L'exemple suivant construit un type **Boolean** nullable et déclare une variable de ce type.

```
Dim ridesBusToWork As Nullable(Of Boolean)
```

La variable `ridesBusToWork` peut contenir la valeur **True**, la valeur **False** ou aucune valeur. Sa valeur par défaut initiale n'est pas définie, ce qui signifie, dans ce cas, que les informations n'ont pas encore été obtenues pour cette personne. E revanche, **False** peut indiquer que les informations ont été obtenues et que la personne ne prend pas le bus pour aller au travail.

Vous pouvez déclarer des variables et des propriétés avec des types nullables, et vous pouvez déclarer un tableau avec des éléments d'un type nullable. Vous pouvez déclarer des procédures avec des types nullables comme paramètres, et vous pouvez retourner un type nullable à partir d'une procédure **Function**.

Vous ne pouvez pas construire un type nullable sur un type référence, par exemple un tableau, une **String** ou une classe. Le type sous-jacent doit être un type valeur. Pour plus d'informations, consultez Types valeur et types référence.

Utilisation d'une variable de type nullable

Les membres les plus importants d'un type nullable sont ses propriétés HasValue et Value. Pour une variable d'un type nullable, **HasValue** vous indique si la variable contient une valeur définie. Si **HasValue** a la valeur **True**, vous pouvez lire la valeur de **Value**. Notez que **HasValue** et **Value** sont des propriétés **ReadOnly**.

Valeurs par défaut

Lorsque vous déclarez une variable avec un type nullable, sa propriété **HasValue** a la valeur par défaut **False**. Cela signifie que la variable proprement dite n'a par défaut aucune valeur définie, et ne prend pas la valeur par défaut de son type valeur sous-jacent. Dans l'exemple suivant, la variable `numberOfChildren` n'a initialement aucune valeur définie, même si la valeur par défaut du type **Integer** est 0.

```
Dim numberOfChildren As Nullable(Of Integer)
```

Une valeur null est utile pour indiquer une valeur non définie ou inconnue. Si `numberOfChildren` a été déclaré comme **Integer**, aucune valeur ne peut indiquer que les informations ne sont pas actuellement disponibles.

Stockage de valeurs

Vous stockez normalement une valeur dans une variable ou une propriété d'un type nullable. L'exemple suivant assigne une valeur à la variable `numberOfChildren` déclarée dans l'exemple précédent.

```
numberOfChildren = 2
```

Si une variable ou une propriété d'un type nullable contient une valeur définie, vous pouvez faire en sorte qu'elle rétablisse son état initial, c'est-à-dire sans valeur assignée. Pour ce faire, affectez **Nothing** à la variable ou la propriété, comme indiqué dans l'exemple suivant.

```
numberOfChildren = Nothing
```

Remarque

Bien que vous puissiez assigner **Nothing** à une variable d'un type nullable, vous ne pouvez pas la tester pour **Nothing**. Vous devez tester sa propriété **HasValue** pour **False**.

Récupération de valeurs

Pour récupérer la valeur d'une variable d'un type nullable, vous devez d'abord tester sa propriété **HasValue** pour confirmer

qu'une valeur lui est assignée. Si vous essayez de lire la valeur lorsque **HasValue** a la valeur **False**, Visual Basic lève une exception InvalidOperationException. L'exemple suivant décrit la méthode recommandée pour lire la variable `numberOfChildren` des exemples précédents.

```
If numberOfChildren.HasValue Then
    MsgBox("There are " & CStr(numberOfChildren) & " children.")
Else
    MsgBox("It is not known how many children there are.")
End If
```

Utilisation de types nullables avec des données

Une base de données est l'un des emplacements les plus importants pour utiliser des types nullables. Tous les objets de base de données ne prennent pas en charge actuellement des types nullables, contrairement aux adaptateurs de table générés par le concepteur. Consultez « Prise en charge des types qui autorisent la valeur Null par le TableAdapter » dans Vue d'ensemble de TableAdapter.

Voir aussi

Tâches

Dépannage des types de données

Référence

InvalidOperationException

Concepts

Types de données en Visual Basic

Types valeur et types référence

Vue d'ensemble de TableAdapter

Autres ressources

Implémentation des types de données

Implémentation des types de données

Les types de données Visual Basic peuvent être classés en fonction du fait qu'une variable d'un certain type enregistre ses propres données ou un pointeur vers les données. Si elle enregistre ses propres données, il s'agit d'un *type valeur* ; si elle contient un pointeur vers les données stockées à un autre endroit de la mémoire, il s'agit d'un *type référence*. Cette classification affecte la manière dont un type de données est implémenté.

Dans cette section

Types valeur et types référence

Présente la différence entre les types valeur et les types référence et explique dans laquelle de ces deux catégories se range chaque type de données.

Les types de données sont des types .NET Framework

Décrit comment les types de données Visual Basic sont implémentés dans le .NET Framework sous-jacent.

Rubriques connexes

Types de données en Visual Basic

Introduit les types de données Visual Basic et décrit comment les utiliser.

Liste des types de données (Visual Basic)

Répertorie les types de données élémentaires disponibles dans Visual Basic.

Types valeur et types référence

Un type de données est un *type valeur* s'il contient des données dans l'espace qui lui est alloué en mémoire. Un *type référence* contient un pointeur vers un autre emplacement en mémoire contenant les données.

Types valeur

Les types valeur incluent les éléments suivants :

- Tous les types de données numériques
- **Boolean**, **Char** et **Date**
- Toutes les structures, même si leurs membres sont des types référence
- Les énumérations, dans la mesure où leur type sous-jacent est toujours **SByte**, **Short**, **Integer**, **Long**, **Byte**, **UShort**, **UInteger** ou **ULong**.

Types référence

Les types référence incluent les éléments suivants :

- **String**
- Tous les tableaux, même si leurs éléments sont des types valeur
- Les types classe, tels que Form.
- Délégués

Éléments qui ne sont pas des types

Les éléments de programmation suivants ne sont pas considérés comme des types car vous ne pouvez les spécifier comme type de données pour un élément déclaré :

- Espaces de noms
- Modules
- Événements
- Propriétés et procédures
- Variables, constantes et champs

Utilisation du type de données Object

Vous pouvez assigner au choix un type référence ou un type valeur à une variable de type de données **Object**. Une variable **Object** contient toujours un pointeur vers les données, jamais les données elles-mêmes. Toutefois, si vous assignez un type valeur à une variable **Object**, elle se comporte comme si elle contenait ses propres données. Pour plus d'informations, consultez Object, type de données.

Vous pouvez déterminer si une variable **Object** agit comme type référence ou type valeur en la passant à la méthode IsReference sur la classe Information dans l'espace de noms Microsoft.VisualBasic. **Microsoft.VisualBasic.Information.IsReference(System.Object)** retourne **True** si le contenu de la variable **Object** représente un type référence.

Voir aussi

Référence

Object, type de données

Concepts

Les types de données sont des types .NET Framework

Utilisation efficace des types de données

Autres ressources

Conversions de type en Visual Basic

Structures constituées à partir de vos propres types de données

Les types de données sont des types .NET Framework

Chaque type de données élémentaire de Visual Basic est pris en charge par une structure ou une classe dans l'espace de noms System. Le compilateur utilise chaque mot clé de type de données comme un alias pour la structure ou la classe sous-jacente. Par exemple, déclarer une variable avec le mot réservé **Byte** équivaut à la déclarer avec le nom de structure qualifié complet System.Byte.

Des classes et des structures supplémentaires sont disponibles dans le .NET Framework.

Le Common Language Runtime (CLR) prend également en charge des structures et des classes qui ne sont pas disponibles dans Visual Basic. Par exemple, la structure System.Guid fournit un identificateur global unique (GUID) et la classe System.TimeZone prend en charge un fuseau horaire. Vous pouvez utiliser ces types pour déclarer des variables et des constantes et vous pouvez accéder aux méthodes que le .NET Framework implémente sur ces types. Toutefois, Visual Basic ne prend pas en charge les opérations ni les conversions de type impliquant des types qu'il ne fournit pas.

Types valeur et types référence

Dans le .NET Framework, une structure est un *type valeur* et une classe est un *type référence*. C'est pourquoi les types valeur, tels que **Char** et **Integer** sont implémentés par les structures du .NET Framework, tandis que les types référence, tels que **Object** et **String** sont pris en charge par les classes du .NET Framework. Notez que chaque tableau est un type référence, même si ses membres sont des types valeur, et que chaque structure est un type valeur, même si elle a des membres de type référence.

Étant donné que chaque type référence représente une classe du .NET Framework sous-jacente, vous devez utiliser le mot clé New (Visual Basic) lorsque vous l'initialisez. L'instruction suivante initialise un tableau.

```
Dim totals() As Single = New Single(8) {}
```

Vous pouvez également utiliser le mot clé **New** pour initialiser un type valeur. Cela s'avère particulièrement utile si le type possède un constructeur qui accepte des paramètres. Le constructeur Decimal en constitue un exemple ; il construit une nouvelle valeur **Decimal** à partir des parties fournies.

Les types de données ont des membres

Étant pris en charge par les structures et les classes du .NET Framework, les types de données Visual Basic ont des *membres*. Ces membres incluent des constructeurs, des méthodes, des propriétés et des champs. Vous pouvez accéder aux membres (à l'exception des constructeurs) d'une variable comme vous accédez aux méthodes et propriétés d'un objet.

L'exemple suivant utilise les propriétés Year, Month et Day et la méthode DaysInMonth de la structure System.DateTime pour déterminer combien de jours il reste dans le mois actuel.

```
Dim current As Date = Now
Dim daysRemaining As Integer
daysRemaining = Date.DaysInMonth(current.Year, current.Month) - current.Day
```

Notez que vous devez qualifier une référence à un membre de type de données avec soit le nom du type (`Date`), soit le nom d'une variable déclarée comme étant de ce type (`current`).

Exemples de membres de types de données

Les exemples de code suivants illustrent certains des champs, propriétés et méthodes qui peuvent être utilisés avec les types de données.

```
<Char>.IsDigit() ' Returns True if character is a numeric digit.

<Char>.IsLower() ' Returns True if character is a lowercase letter.

<Date>.IsLeapYear() ' Returns True if current year is a leap year.

<Date>.ToUniversalTime() ' Returns local date/time converted to UTC.

<Double>.IsInfinity() ' Returns True if contents represent infinity.

<Double>.IsNaN() ' Returns True if contents are not a number (0/0).

<Long>.MaxValue ' Constant representing largest positive Int64 value.
```

```
<Object>.GetType() ' Returns Type object representing type of <Object>.

<Object>.GetType().GetTypeCode() ' Returns type code of <Object>.

<String>.Chars(<index>) ' Character at position <index> of <String>.

<String>.Length ' Number of characters currently in <String>.
```

Tous les types numériques, y compris **Byte** et **Char**, exposent les champs publics MaxValue et MinValue, ce qui peut s'avérer très utile lors de l'utilisation de ces types de données.

L'équivalence des membres de types de données n'est pas assurée

Le .NET Framework fournit également un certain nombre de méthodes sur des types de données qui peuvent sembler équivalentes à des fonctions et des mots clés Visual Basic. Toutefois, Visual Basic n'utilise pas toujours les méthodes du .NET Framework pour accomplir la conversion ou d'autres opérations et les résultats ne sont pas toujours identiques.

Par exemple, la méthode ToSingle effectue le même type d'action que le mot clé **CSng** sur une expression **Decimal**. Mais il n'est pas garanti que **CSng** utilise **System.**<*data type*>**.ToSingle** et il n'est donc pas certain que les résultats soient identiques dans des conditions marginales.

En général, vous devez utiliser les éléments de programmation Visual Basic car ils sont plus faciles à utiliser et rendent votre code plus lisible. Dans certains cas, vous pourrez avoir besoin de certaines fonctionnalités supplémentaires fournies par une méthode du .NET Framework. Pour obtenir un exemple, consultez « L'opérateur Mod ne retourne pas de résultats exacts » dans Dépannage des types de données.

Voir aussi

Référence

Boolean
Byte
Char
DateTime
Decimal
Double
Guid
TimeZone

Concepts

Types valeur et types référence
Structures et classes

Conversions de type en Visual Basic

L'opération consistant à remplacer une valeur d'un certain type de données par une valeur d'un autre type est appelée *conversion*. Les conversions sont soit *étendues*, soit *restrictives*, en fonction de la capacité de données des types concernés. Elles sont également *implicites* ou *explicites*, en fonction de la syntaxe du code source.

Dans cette section

Conversions étendues et restrictives

Le fait que le type de destination puisse ou non contenir les données détermine la classification de la conversion.

Conversions implicites et explicites

Le fait que Visual Basic l'effectue automatiquement ou non détermine également la classification de la conversion.

Modification des valeurs lors de la conversion

Explique en quoi la conversion dans un autre type de données peut modifier la représentation de la valeur ou des données.

Conversion entre des chaînes et d'autres types

Décrit la conversion entre des chaînes et des valeurs numériques, **Boolean** ou date/heure.

Comment : convertir un objet en un autre type dans Visual Basic

Explique comment convertir une variable **Object** en tout autre type de données.

Conversion des tableaux

Vous guide pas à pas dans le processus de conversion entre des tableaux de types de données différents.

Rubriques connexes

Types de données en Visual Basic

Introduit les types de données Visual Basic et décrit comment les utiliser.

Liste des types de données (Visual Basic)

Répertorie les types de données élémentaires disponibles dans Visual Basic.

Dépannage des types de données

Présente des problèmes courants qui peuvent survenir lors de l'utilisation des types de données.

Conversions étendues et restrictives

Le fait que le résultat se situe ou non dans la plage du type de données de destination constitue un élément d'information important dans la conversion de type. Dans une *conversion étendue*, la valeur est convertie dans un type de données capable d'accueillir toutes les valeurs possibles des données d'origine. Dans une *conversion restrictive*, la valeur est convertie dans un type de données susceptible de ne pas pouvoir accueillir certaines valeurs possibles des données.

Conversions étendues

Le tableau suivant répertorie les conversions étendues standard.

Type de données	Extension aux types de données [1]
SByte	**SByte**, **Short**, **Integer**, **Long**, **Decimal**, **Single**, **Double**
Byte	**Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single**, **Double**
Short	**Short**, **Integer**, **Long**, **Decimal**, **Single**, **Double**
UShort	**UShort**, **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single**, **Double**
Integer	**Integer**, **Long**, **Decimal**, **Single**, **Double**[2]
UInteger	**UInteger**, **Long**, **ULong**, **Decimal**, **Single**, **Double** [2]
Long	**Long**, **Decimal**, **Single**, **Double** [2]
ULong	**ULong**, **Decimal**, **Single**, **Double** [2]
Decimal	**Decimal**, **Single**, **Double** [2]
Single	**Single**, **Double**
Double	**Double**
Tout type énuméré (Enum)	Son type intégral sous-jacent et tout type auquel le type sous-jacent s'étend.
Char	**Char**, **String**
Tableau **Char**	**Char** (tableau), **String**
Tout type	Objet
Tout type dérivé	Tout type de base dont il est dérivé [3]
Tout type	Toute interface qu'il implémente.
Nothing	Tout type de données ou type d'objet.

[1] Par définition, chaque type de données s'élargit par lui-même.

[2] Les conversions de **Integer**, **UInteger**, **Long**, **ULong** ou **Decimal** en **Single** ou **Double** peuvent entraîner un manque de précision, mais jamais une perte d'amplitude. Elles n'entraînent donc jamais de perte d'information.

[3] Il peut sembler surprenant qu'une conversion entre un type dérivé et l'un de ses types de base soit étendue. Cela s'explique

par le fait que le type dérivé contient tous les membres du type de base ; il se qualifie donc comme une instance du type de base. Dans le sens opposé, le type de base ne contient aucun nouveau membre défini par le type dérivé.

Les conversions étendues réussissent toujours au moment de l'exécution et sans jamais aucune perte de données. Vous pouvez toujours les exécuter implicitement, que l'instruction Option Strict attribue la valeur **On** ou **Off** au commutateur de vérification de type.

Conversions restrictives

Les conversions restrictives standard sont les suivantes :

- Conversions inverses des conversions étendues dans le tableau précédent (excepté que chaque type s'élargit par lui-même).
- Conversions entre un type Boolean et tout type numérique (dans l'un et l'autre sens).
- Conversions de n'importe quel type numérique en un type énuméré (**Enum**).
- Conversions dans l'un et l'autre sens entre String et tout type numérique, **Boolean** ou Date.
- Conversions entre un type de données ou un type d'objet et un type dérivé de lui.

Les conversions restrictives ne réussissent pas toujours au moment de l'exécution et peuvent échouer ou entraîner la perte de données. Une erreur se produit si le type de données de destination ne peut pas accueillir la valeur convertie. Par exemple, une conversion numérique peut provoquer un dépassement de capacité. Le compilateur ne vous permet pas d'exécuter des conversions restrictives implicitement sauf si l'instruction Option Strict attribue la valeur **Off** au commutateur de vérification de type.

Utilisation des conversions restrictives

Utilisez une conversion restrictive lorsque vous savez que la valeur source peut être convertie dans le type de données de destination sans erreur ni perte de données. Par exemple, si vous avez un type **String** dont vous savez qu'il contient soit "True", soit "False", vous pouvez utiliser le mot clé **CBool** pour le convertir en **Boolean**.

Exceptions pendant la conversion

Étant donné que les conversions étendues aboutissent toujours, elles ne lèvent pas d'exceptions. Les conversions restrictives, lorsqu'elles échouent, lèvent le plus souvent les exceptions suivantes :

- InvalidCastException — si aucune conversion n'est définie entre les deux types
- OverflowException — (types intégraux uniquement) si la valeur convertie est trop grande pour le type cible

Si une classe ou une structure définit CType Function pour qu'il serve d'opérateur de conversion vers ou depuis cette classe ou cette structure, **CType** peut lever n'importe quelle exception qu'il estime appropriée. De plus, **CType** peut appeler des fonctions Visual Basic ou des méthodes .NET Framework qui peuvent à leur tour lever diverses exceptions.

Voir aussi

Tâches

Comment : convertir un objet en un autre type dans Visual Basic

Référence

Liste des types de données (Visual Basic)

Fonctions de conversion de types de données

Concepts

Types de données en Visual Basic

Conversions implicites et explicites

Modification des valeurs lors de la conversion

Conversion entre des chaînes et d'autres types

Conversion des tableaux

Programmation sans type dans Visual Basic

Autres ressources

Conversions de type en Visual Basic

Conversions implicites et explicites

Une *conversion implicite* ne requiert pas de syntaxe particulière dans le code source. Dans l'exemple suivant, Visual Basic convertit implicitement la valeur de `k` en une valeur à virgule flottante simple précision avant de l'assigner à `q`.

```
Dim k As Integer
Dim q As Double
' Integer widens to Double, so you can do this with Option Strict On.
k = 432
q = K
```

Une *conversion explicite* utilise un mot clé de conversion de type. Visual Basic fournit plusieurs mots clé à cet effet, qui forcent une expression entre parenthèses au type de données souhaité. Ces mots clés agissent comme des fonctions, mais le compilateur génère le code en ligne, si bien que l'exécution est légèrement plus rapide qu'avec un appel de fonction.

Dans l'exemple ci-dessous qui fait suite à l'exemple précédent, le mot clé **CInt** reconvertit la valeur de `q` en un entier avant de l'assigner à `k`.

```
' q had been assigned the value 432 from k.
q = Math.Sqrt(q)
k = CInt(q)
' k now has the value 21 (rounded square root of 432).
```

Mots clés de conversion

Le tableau suivant répertorie les mots clés de conversion disponibles.

Mot clé de conversion de type	Convertit une expression vers le type de données	Types de données autorisés d'expression à convertir
CBool	Boolean, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **String**, **Object**
CByte	Byte, type de données (Visual Basic)	Tout type numérique (y compris **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CChar	Char, type de données (Visual Basic)	**String**, **Object**
CDate	Type de données date (Visual Basic)	**String**, **Object**
CDbl	Double, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CDec	Decimal, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CInt	Integer, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CLng	Long, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CObj	Object, type de données	Tout type
CSByte	SByte, type de données (Visual Basic)	Tout type numérique (y compris **Byte** et les types énumérés), **Boolean**, **String**, **Object**

CShort	Short, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CSng	Single, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CStr	String, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, tableau **Char**, **Char**, **Date**, **Object**
CType	Type spécifié après la virgule (**,**)	Lors d'une conversion vers un *type de données élémentaire* (y compris un tableau d'un type élémentaire), les mêmes types que ceux qui sont autorisés pour le mot clé de conversion correspondant Lors d'une conversion vers un *type de données composite*, les interfaces qu'il implémente et les classes à partir desquelles il hérite Lors de la conversion vers une classe ou une structure sur laquelle vous avez surchargé **CType**, cette classe ou structure
CUInt	UInteger, type de données	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CULng	ULong, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**
CUShort	UShort, type de données (Visual Basic)	Tout type numérique (y compris **Byte**, **SByte** et les types énumérés), **Boolean**, **String**, **Object**

La fonction CType

La fonction CType opère sur deux arguments. Le premier correspond à l'expression à convertir, le second au type de données de destination ou à la classe d'objet. Notez que le premier argument doit être une expression, et non un type.

CType est une *fonction inline*, ce qui signifie que le code compilé effectue la conversion, souvent sans générer d'appel de fonction. Les performances sont alors améliorées.

Pour obtenir une comparaison entre **CType** et les autres mots clés de conversion de type, consultez DirectCast et TryCast.

Types élémentaires

L'exemple suivant illustre l'utilisation du mot clé **CType** :

```
k = CType(q, Integer)
' The following statement coerces w to the specific object class Label.
f = CType(w, Label)
```

Types composites

Vous pouvez utiliser le mot clé **CType** pour convertir des valeurs vers des types de données composites ou élémentaires. Vous pouvez également l'utiliser pour forcer la conversion d'une classe d'objet vers le type de l'une de ses interfaces, comme dans l'exemple suivant.

```
' Assume class cZone implements interface iZone.
Dim h As Object
' The first argument to CType must be an expression, not a type.
Dim cZ As cZone
' The following statement coerces a cZone object to its interface iZone.
h = CType(cZ, iZone)
```

Types tableau

CType peut également convertir des types de données tableau, comme dans l'exemple suivant.

```
Dim v() As classV
Dim obArray() As Object
' Assume some object array has been assigned to obArray.
' Check for run-time type compatibility.
If TypeOf obArray Is classV()
    ' obArray can be converted to classV.
    v = CType(obArray, classV())
End If
```

Pour obtenir des informations supplémentaires et un exemple, consultez Conversion des tableaux.

Types définissant CType

Vous pouvez définir **CType** sur une classe ou une structure que vous avez définie. Cela vous permet de convertir des valeurs vers et à partir du type de votre classe ou structure. Pour obtenir des informations supplémentaires et un exemple, consultez Comment : définir un opérateur de conversion.

Remarque

Les valeurs utilisées avec un mot clé de conversion doivent être acceptées par le type de données de destination, sans quoi u ne erreur se produit. Par exemple, si vous tentez de convertir un type **Long** en **Integer**, la valeur du type **Long** doit se situer dans la plage acceptée par le type de données **Integer**.

Attention

La spécification de **CType** pour effectuer une conversion à partir d'un type de classe vers un autre échoue au moment de l'ex écution si le type source ne dérive pas du type de destination. Un tel échec lève une exception InvalidCastException.

Toutefois, si l'un des types est une structure ou une classe que vous avez définie et si vous avez défini **CType** sur cette structure ou cette classe, une conversion peut réussir si elle satisfait aux exigences de votre **CType**. Consultez Comment : définir un opérateur de conversion.

Pour désigner le processus de conversion explicite, on parle également de *casting* d'une expression vers un certain type de données ou une classe d'objet.

Voir aussi

Tâches

Comment : convertir un objet en un autre type dans Visual Basic
Dépannage des types de données

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données

Concepts

Modification des valeurs lors de la conversion
Conversion entre des chaînes et d'autres types
Programmation sans type dans Visual Basic

Autres ressources

Conversions de type en Visual Basic
Structures constituées à partir de vos propres types de données

Modification des valeurs lors de la conversion

La conversion à partir d'un type valeur enregistre une copie de la valeur source dans la destination de la conversion. Toutefois, cette copie n'est pas une image fidèle de la valeur source. Le type de données de destination enregistre différemment les valeurs ; il peut même arriver que la valeur représentée change en fonction du type de conversion effectuée.

Modification lors des conversions étendues et restrictives

Les *conversions restrictives* changent la copie de destination de la valeur source en entraînant éventuellement une perte d'information. Par exemple, une valeur fractionnaire est arrondie lorsqu'elle est convertie en type intégral et un type numérique converti en **Boolean** se trouve réduit à la valeur **True** ou **False**.

Les *conversions étendues* préservent la valeur source, mais peuvent modifier sa représentation. Cela se produit notamment si vous convertissez un type intégral en **Decimal** ou de **Char** en **String**.

La valeur source d'origine n'est pas modifiée à la suite d'une conversion.

Modification lors des conversions de type référence

Une conversion à partir d'un *type référence* ne copie que le pointeur vers la valeur. La valeur elle-même n'est ni copiée ni modifiée. Seul peut être modifié le type de données de la variable contenant le pointeur. Dans l'exemple suivant, le type de données est converti de la classe dérivée vers sa classe de base, mais l'objet vers lequel pointent les deux variables reste inchangé.

```
' Assume class cSquare inherits from class cShape.
Dim shape As cShape
Dim square As cSquare = New cSquare
' The following statement performs a widening
' conversion from a derived class to its base class.
shape = square
```

Voir aussi

Tâches

Comment : convertir un objet en un autre type dans Visual Basic

Référence

Liste des types de données (Visual Basic)

Fonctions de conversion de types de données

Concepts

Conversions étendues et restrictives

Conversions implicites et explicites

Conversion entre des chaînes et d'autres types

Conversion des tableaux

Programmation sans type dans Visual Basic

Autres ressources

Conversions de type en Visual Basic

Structures constituées à partir de vos propres types de données

Conversion entre des chaînes et d'autres types

Vous pouvez convertir une valeur numérique, **Boolean** ou date/heure vers le type **String**. Vous pouvez également opérer la conversion inverse, c'est-à-dire d'une chaîne vers un type de données numérique, **Boolean** ou **Date**, à condition que le contenu de la chaîne soit une valeur reconnue par le type de données de destination. Si tel n'est pas le cas, une erreur d'exécution se produit.

Ces différentes conversions (dans l'un et l'autre sens) sont des conversions restrictives. Utilisez les mots clés de conversion de type (**CBool**, **CByte**, **CDate**, **CDbl**, **CDec**, **CInt**, **CLng**, **CSByte**, **CShort**, **CSng**, **CStr**, **CUInt**, **CULng**, **CUShort** et **CType**). La fonction Format et la fonction Val vous permettent de mieux contrôler les conversions entre les chaînes et les nombres.

Si vous avez défini une classe ou une structure, vous pouvez définir des opérateurs de conversion de type entre **String** et le type de votre classe ou structure. Pour plus d'informations, consultez Comment : définir un opérateur de conversion.

Conversion de nombres en chaînes

Vous pouvez utiliser la fonction **Format** pour convertir un nombre en chaîne mise en forme, dotée non seulement des chiffres appropriés mais également de symboles de format tels que la devise (**$**, par exemple), de séparateurs des milliers ou de *symboles de groupement des chiffres* (**,**, par exemple), et d'un séparateur décimal (**.**, par exemple). **Format** utilise automatiquement les symboles correspondant aux paramètres d'**Options régionales** spécifiés dans le **Panneau de configuration** de Windows.

Notez que l'opérateur de concaténation (**&**) peut convertir implicitement un nombre en chaîne, comme le montre l'exemple suivant.

```
' The following statement converts count to a String value.
Str = "The total count is " & count
```

Conversion de chaînes en nombres

Vous pouvez utiliser la fonction **Val** pour convertir explicitement les chiffres d'une chaîne en un nombre. La fonction **Val** lit la chaîne jusqu'à ce qu'elle rencontre un caractère autre qu'un chiffre, un espace, une tabulation, un saut de ligne ou un point. Les séquences "&O" et "&H" altèrent la base du système numérique et interrompent la lecture. Tant qu'elle n'arrête pas sa lecture, la fonction **Val** convertit tous les caractères appropriés en une valeur numérique. Par exemple, l'instruction suivante retourne la valeur `141.825`.

```
Val(" 14 1.825 miles")
```

Lorsque Visual Basic convertit une chaîne en valeur numérique, il utilise les **Options régionales** spécifiées dans le **Panneau de configuration** de Windows pour interpréter le séparateur de milliers, le séparateur décimal et le symbole monétaire. Cela signifie qu'une conversion peut réussir avec certains paramètres, mais pas avec d'autres. Par exemple, la valeur `"$14.20"` est acceptée pour le paramètre régional Anglais (États-Unis), mais pas pour un paramètre régional français.

Voir aussi

Tâches

Comment : convertir un objet en un autre type dans Visual Basic

Référence

Liste des types de données (Visual Basic)

Fonctions de conversion de types de données

Concepts

Conversions étendues et restrictives

Conversions implicites et explicites

Modification des valeurs lors de la conversion

Conversion des tableaux

Introduction aux applications internationales basées sur le .NET Framework

Autres ressources

Conversions de type en Visual Basic

Comment : convertir un objet en un autre type dans Visual Basic

Vous pouvez convertir une variable **Object** en un autre type de données en utilisant un mot clé de conversion, tel que CType Function.

Exemple

L'exemple suivant convertit une variable **Object** en un **Integer** et un **String**.

```
Public Sub objectConversion(ByVal anObject As Object)
    Dim anInteger As Integer
    Dim aString As String
    anInteger = CType(anObject, Integer)
    aString = CType(anObject, String)
End Sub
```

Si vous savez que le contenu d'une variable **Object** est d'un type de données particulier, il est préférable de convertir la variable en ce type de données. Si vous continuez à utiliser la variable **Object**, vous occasionnez une *conversion boxing*, une *conversion unboxing* (pour un type valeur) ou une *liaison tardive* (pour un type référence). Ces opérations nécessitent toutes un temps d'exécution supplémentaire et ralentissent les performances.

Compilation du code

Cet exemple nécessite :

- Référence à l'espace de noms System.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données
Object

Concepts

Conversions étendues et restrictives
Conversions implicites et explicites
Modification des valeurs lors de la conversion
Conversion entre des chaînes et d'autres types
Conversion des tableaux
Programmation sans type dans Visual Basic

Autres ressources

Conversions de type en Visual Basic
Structures constituées à partir de vos propres types de données

Conversion des tableaux

Vous pouvez convertir un type tableau vers un autre type, pourvu que les conditions suivantes soient remplies :

- **Rang égal.** Les rangs des deux tableaux doivent être identiques ; autrement dit, les deux tableaux doivent avoir le même nombre de dimensions. Toutefois, il n'est pas obligatoire que la longueur des dimensions respectives soit identique.
- **Type de données des éléments.** Les types de données des éléments des deux tableaux doivent être des types référence. Vous ne pouvez pas convertir un tableau de type **Integer** en tableau **Long** ou même **Object**, parce que cette opération fait intervenir au moins un type valeur. Pour plus d'informations, consultez Types valeur et types référence.
- **Convertibilité.** Une conversion, qu'elle soit restrictive ou étendue, doit être possible entre les types d'éléments des deux tableaux. Un exemple qui ne satisfait pas à cette condition est une tentative de conversion entre un tableau **String** et un tableau d'une classe dérivée de System.Attribute. Ces deux types n'ont rien en commun et aucune conversion n'existe entre eux.

Les conversions entre deux types tableau sont soit étendues, soit restrictives, selon qu'elles élargissent ou rétrécissent leurs éléments respectifs. Pour plus d'informations, consultez Conversions étendues et restrictives.

Conversion vers un tableau de type Object

Lorsque vous déclarez un tableau **Object**, son type d'élément reste **Object** tant qu'il n'est pas initialisé. Lorsque vous le configurez en tableau de classe particulière, il prend le type de cette classe. Toutefois, son type sous-jacent reste **Object** et vous pouvez ensuite le configurer en un autre tableau de classe non apparentée. Étant donné que toutes les classes sont dérivées du type **Object**, vous pouvez modifier le type d'élément du tableau en remplaçant sa classe par toute autre classe.

Dans l'exemple ci-dessous, aucune conversion n'existe entre les types `student` et **String**, mais ils sont tous deux dérivés du type **Object** et, par conséquent, toutes les assignations sont valides.

```
' Assume student has already been defined as a class.
Dim testArray() As Object
' testArray is still an Object array at this point.
Dim names() As String = New String(3) {"Name0", "Name1", "Name2", "Name3"}
testArray = New student(3) {}
' testArray is now of type student().
testArray = names
' testArray is now a String array.
```

Type sous-jacent d'un tableau

Si vous déclarez à l'origine un tableau avec une classe particulière, son type d'élément sous-jacent correspond à cette classe. Par la suite, si vous le configurez en tableau de classe différente, une conversion doit être effectuée entre les deux classes.

Dans l'exemple ci-après, `students` représente un tableau `student`. Comme aucune conversion n'existe entre **String** et `student`, la dernière instruction se solde par un échec.

```
Dim students() As student
Dim names() As String = New String(3) {"Name0", "Name1", "Name2", "Name3"}
students = New Student(3) {}
' The following statement fails at compile time.
students = names
```

Voir aussi

Tâches

Comment : convertir un objet en un autre type dans Visual Basic

Référence

Liste des types de données (Visual Basic)

Fonctions de conversion de types de données

Concepts

Types de données en Visual Basic

Conversions implicites et explicites

Modification des valeurs lors de la conversion

Conversion entre des chaînes et d'autres types
Programmation sans type dans Visual Basic

Autres ressources

Conversions de type en Visual Basic
Tableaux dans Visual Basic

Structures constituées à partir de vos propres types de données

Une *structure* est une généralisation du type défini par l'utilisateur (UDT, User-Defined Type) des précédentes versions de Visual Basic. En plus des champs, les structures peuvent exposer des propriétés, des méthodes et des événements. Une structure peut implémenter une ou plusieurs interfaces et vous pouvez déclarer des niveaux d'accès propres à chaque champ.

Vous pouvez combiner des éléments de données de différents types pour constituer une structure. Une structure associe un ou plusieurs *éléments* les uns avec les autres et avec la structure elle-même. Lorsque vous déclarez une structure, elle devient un *type de données composite* et vous pouvez déclarer des variables de ce type.

Les structures sont utiles lorsque vous voulez qu'une même variable contienne plusieurs éléments d'information connexes. Par exemple, vous pouvez conserver ensemble le nom, le poste téléphonique et le salaire d'un employé. Vous pouvez enregistrer ces informations dans plusieurs variables ou définir une structure et l'utiliser comme une variable d'employé unique. Les avantages des structures deviennent évidents lorsque vous avez de nombreux employés et par conséquent de nombreuses instances de la variable.

Dans cette section

Comment : déclarer une structure

Explique comment déclarer une structure et ses éléments.

Variables de structure

Explique comment assigner une structure à une variable et accéder à ses éléments.

Structures et autres éléments de programmation

Récapitule comment les structures interagissent avec les tableaux, les objets, les procédures et les unes avec les autres.

Structures et classes

Décrit les similitudes et les différences entre les structures et les classes.

Rubriques connexes

Types de données en Visual Basic

Introduit les types de données Visual Basic et décrit comment les utiliser.

Liste des types de données (Visual Basic)

Répertorie les types de données élémentaires disponibles dans Visual Basic.

Comment : déclarer une structure

Vous commencez une déclaration de structure par l'instruction Structure et vous la terminez par l'instruction **End Structure**. Entre ces deux instructions, vous devez déclarer au moins un *élément*. Les éléments peuvent être de tout type de données, mais au moins un d'entre eux doit être une variable non partagée ou un événement non partagé et non personnalisé.

Vous ne pouvez initialiser aucun des éléments de structure dans la déclaration de structure. Lorsque vous déclarez une variable comme étant d'un type structure, vous assignez des valeurs aux éléments en y accédant par l'intermédiaire de la variable.

Pour une discussion des différences entre les structures et les classes, consultez Structures et classes.

À des fins de démonstration, considérez une situation dans laquelle vous souhaitez assurer le suivi du nom, du poste téléphonique et du salaire d'un employé. Une structure vous permet d'effectuer cette opération dans une seule variable.

Pour déclarer une structure

1. Créez les instructions de début et de fin de la structure.

   Vous pouvez spécifier le niveau d'accès d'une structure à l'aide du mot clé Public (Visual Basic), Protected (Visual Basic), Friend (Visual Basic) ou Private (Visual Basic), ou vous pouvez accepter le paramètre par défaut, **Public**.

   ```
   Private Structure employee
   End Structure
   ```

2. Ajoutez des éléments au corps de la structure.

   Une structure doit avoir au moins un élément. Vous devez déclarer chaque élément et spécifier pour lui un niveau d'accès. Si vous utilisez l'Dim, instruction (Visual Basic) sans mot clé, l'accessibilité prend la valeur par défaut, **Public**.

   ```
   Private Structure employee
       Public givenName As String
       Public familyName As String
       Public phoneExtension As Long
       Private salary As Decimal
       Public Sub giveRaise(raise As Double)
           salary *= raise
       End Sub
       Public Event salaryReviewTime()
   End Structure
   ```

   Le champ `salary` dans l'exemple précédent est **Private**, ce qui signifie qu'il est inaccessible à l'extérieur de la structure, même à partir de la classe conteneur. Toutefois, la procédure `giveRaise` étant de type **Public**, elle peut être appelée de l'extérieur de la structure. De même, vous pouvez déclencher l'événement `salaryReviewTime` depuis l'extérieur de la structure.

   En plus des variables, des procédures **Sub** et des événements, vous pouvez également définir des constantes, des procédures **Function** et des propriétés dans une structure. Vous pouvez désigner au plus une propriété comme *propriété par défaut*, à condition qu'elle accepte au moins un argument. Vous pouvez gérer un événement avec une procédure **Sub** Shared (Visual Basic). Pour plus d'informations, consultez Comment : déclarer et appeler une propriété par défaut en Visual Basic.

Voir aussi

Tâches

Dépannage des types de données

Référence

Type de données défini par l'utilisateur

Concepts

Types de données en Visual Basic

Types de données composites

Variables de structure

Structures et autres éléments de programmation
Structures et classes
Autres ressources
Types de données élémentaires
Implémentation des types de données
Structures constituées à partir de vos propres types de données

Variables de structure

Une fois que vous avez créé une structure, vous pouvez déclarer des variables du niveau procédure ou du niveau module avec ce type. Par exemple, vous pouvez créer une structure qui enregistre des informations à propos d'un ordinateur. C'est ce qu'illustre l'exemple suivant.

```
Public Structure systemInfo
    Public cPU As String
    Public memory As Long
    Public purchaseDate As Date
End Structure
```

Vous pouvez désormais déclarer des variables de ce type. L'exemple suivant illustre ce comportement.

```
Dim mySystem, yourSystem As systemInfo
```

Remarque

Dans les classes et les modules, les structures déclarées à l'aide de Dim, instruction (Visual Basic) ont par défaut un accès public. Si vous souhaitez qu'une structure soit privée, veillez à la déclarer à l'aide du mot clé Private (Visual Basic).

Accès aux valeurs des structures

Pour assigner et extraire des valeurs à partir des éléments d'une variable de structure, vous utilisez la même syntaxe que pour définir et récupérer les propriétés d'un objet. Placez l'opérateur d'accès aux membres (.) entre le nom de la variable de structure et le nom de l'élément. L'exemple suivant accède à des éléments des variables déclarées précédemment comme type `systemInfo`.

```
mySystem.cPU = "486"
Dim tooOld As Boolean
If yourSystem.purchaseDate < #1/1/1992# Then tooOld = True
```

Assigner des variables de structure

Vous pouvez également assigner une variable à une autre si toutes deux ont le même type structure. Cette opération copie tous les éléments d'une structure dans les éléments correspondants d'une autre. L'exemple suivant illustre ce comportement.

```
yourSystem = mySystem
```

Si un élément de structure est un type référence (par exemple, **String**, **Object** ou un tableau), le pointeur vers les données est copié. Dans l'exemple précédent, si `systemInfo` avait inclus une variable d'objet, l'exemple précédent aurait copié le pointeur à partir de `mySystem` dans `yourSystem`, et une modification vers les données de l'objet serait appliquée par l'intermédiaire d'une structure lors de son accès par l'autre structure.

Voir aussi

Tâches

Dépannage des types de données
Comment : déclarer une structure

Référence

Structure, instruction

Concepts

Types de données en Visual Basic
Types de données composites
Structures et autres éléments de programmation
Structures et classes

Autres ressources

Types de données élémentaires
Implémentation des types de données

Structures constituées à partir de vos propres types de données

Structures et autres éléments de programmation

Vous pouvez utiliser des structures avec des tableaux, des objets, des procédures, mais aussi les unes avec les autres. Les interactions utilisent la même syntaxe que ces éléments pris individuellement.

Remarque

Vous ne pouvez initialiser aucun élément de structure dans la déclaration de structure. Vous pouvez assigner des valeurs uniquement aux éléments d'une variable qui a été déclarée comme un type structure.

Structures et tableaux

Une structure peut contenir un tableau comme un ou plusieurs de ses éléments. L'exemple suivant illustre ce comportement :

```
Public Structure systemInfo
    Public cPU As String
    Public memory As Long
    Public diskDrives() As String
    Public purchaseDate As Date
End Structure
```

Pour accéder aux valeurs d'un tableau dans une structure, procédez comme pour accéder à une propriété d'un objet. L'exemple suivant illustre ce comportement :

```
Dim mySystem As systemInfo
ReDim mySystem.diskDrives(3)
mySystem.diskDrives(0) = "1.44 MB"
```

Vous pouvez également déclarer un tableau de structures. L'exemple suivant illustre ce comportement :

```
Dim allSystems(100) As systemInfo
```

Suivez les mêmes règles pour accéder aux composants de cette architecture de données. L'exemple suivant illustre ce comportement :

```
ReDim allSystems(5).diskDrives(3)
allSystems(5).CPU = "386SX"
allSystems(5).diskDrives(2) = "100M SCSI"
```

Structures et objets

Une structure peut contenir un objet comme un ou plusieurs de ses éléments. L'exemple suivant illustre ce comportement :

```
Protected Structure userInput
    Public userName As String
    Public inputForm As System.Windows.Forms.Form
    Public userFileNumber As Integer
End Structure
```

Dans une déclaration de ce type, utilisez une classe d'objet spécifique de préférence à **Object**.

Structures et procédures

Vous pouvez passer une structure comme un argument de procédure. L'exemple suivant illustre ce comportement :

```
Public currentCPUName As String = "700MHz Pentium compatible"
Public currentMemorySize As Long = 256
Public Sub fillSystem(ByRef someSystem As systemInfo)
    someSystem.cPU = currentCPUName
```

```
    someSystem.memory = currentMemorySize
    someSystem.purchaseDate = Now
End Sub
```

Dans l'exemple précédent, la structure est passée *par référence*, ce qui permet à la procédure de modifier ses éléments de sorte que les modifications prennent effet dans le code appelant. Pour protéger une structure contre de telles modifications, passez-la par valeur.

Vous pouvez également retourner une structure d'une procédure **Function**. L'exemple suivant illustre ce comportement :

```
Dim allSystems(100) As systemInfo
Function findByDate(ByVal searchDate As Date) As systemInfo
    Dim i As Integer
    For i = 1 To 100
        If allSystems(i).purchaseDate = searchDate Then Return allSystems(i)
    Next i
   ' Process error: system with desired purchase date not found.
End Function
```

Structures dans d'autres structures

Les structures peuvent contenir d'autres structures. L'exemple suivant illustre ce comportement :

```
Public Structure driveInfo
    Public type As String
    Public size As Long
End Structure
Public Structure systemInfo
    Public cPU As String
    Public memory As Long
    Public diskDrives() As driveInfo
    Public purchaseDate As Date
End Structure
Dim allSystems(100) As systemInfo
ReDim allSystems(1).diskDrives(3)
allSystems(1).diskDrives(0).type = "Floppy"
```

Vous pouvez également utiliser cette technique pour encapsuler une structure définie dans un module à l'intérieur d'une autre structure définie dans un module différent.

Les structures peuvent s'imbriquer dans d'autres structures sans limite de profondeur.

Voir aussi

Tâches

Dépannage des types de données
Comment : déclarer une structure

Référence

Structure, instruction

Concepts

Types de données en Visual Basic
Types de données composites
Variables de structure
Structures et classes

Autres ressources

Types de données élémentaires
Implémentation des types de données
Structures constituées à partir de vos propres types de données

Structures et classes

Visual Basic unifie la syntaxe des structures et des classes, en conséquence de quoi les deux types d'entités prennent en charge les mêmes fonctionnalités. Toutefois, il traite également des principales différences existant entre les classes et les structures.

Les classes ont l'avantage d'être des types référence (il est plus efficace de passer une référence qu'une variable de structure avec toutes ses données). D'un autre côté, les structures n'exigent pas d'allocation de mémoire dans le tas global.

Parce que vous ne pouvez pas hériter d'elles, les structures ne doivent être utilisées que pour les objets qui n'ont pas besoin d'être étendus. Utilisez les structures lorsque l'objet que vous souhaitez créer a une taille d'instance réduite et tenez compte des caractéristiques de performances des classes par rapport à celles des structures.

Similitudes

Les structures et les classes sont identiques à plusieurs égards :

- elles représentent des types *conteneurs*, ce qui signifie qu'elles peuvent contenir d'autres types comme membres ;
- elles ont des membres qui peuvent comprendre des constructeurs, des méthodes, des propriétés, des champs, des constantes, des énumérations, des événements et des gestionnaires d'événements ; Toutefois, ne confondez pas ces membres avec les *éléments* déclarés d'une structure ;
- leurs membres peuvent présenter des niveaux d'accès individualisés. Par exemple, un membre peut être déclaré **Public** et un autre **Private** ;
- elles peuvent implémenter des interfaces ;
- elles peuvent avoir des constructeurs partagés, avec ou sans paramètres.
- elles peuvent exposer une *propriété par défaut*, à condition que celle-ci accepte au moins un paramètre ;
- elles peuvent déclarer et déclencher des événements, et déclarer des délégués.

Différences

Les structures et les classes diffèrent à plusieurs égards :

- les structures sont des *types valeur* ; les classes sont des *types référence*. Une variable d'un type structure contient les données de la structure, plutôt qu'une référence aux données comme le fait un type classe ;
- les structures utilisent l'allocation de la pile, les classes l'allocation de tas ;
- tous les éléments de structure sont **Public** par défaut, les variables et les constantes de classe sont **Private** par défaut, tandis que les autres membres de classe sont **Public** par défaut. Ce comportement pour les membres de classe assure la compatibilité avec le système de valeurs par défaut Visual Basic 6.0 ;
- une structure doit contenir au moins une variable non partagée ou un élément événement non personnalisé et non partagé, alors qu'une classe peut être entièrement vide ;
- les éléments de structure ne peuvent pas être déclarés comme **Protected**, contrairement aux membres de classe ;
- une procédure de structure ne peut gérer des événements que s'il s'agit d'une procédure **Sub** Shared (Visual Basic) et qu'elle utilise l'instruction AddHandler, alors que toute procédure de classe peut gérer des événements avec le mot clé Handles ou l'instruction **AddHandler**. Pour plus d'informations, consultez Événements et gestionnaires d'événements.
- les déclarations de variable de structure ne peuvent pas spécifier d'initialiseurs ni les tailles initiales des tableaux, contrairement aux déclarations de variable de classe ;
- les structures héritent implicitement de la classe System.ValueType et ne peuvent pas hériter d'un autre type, tandis que les classes peuvent hériter d'une ou plusieurs classes autres que **System.ValueType** ;
- il est possible d'hériter des classes, mais pas des structures ;
- les structures n'étant jamais arrêtées, le Common Language Runtime (CLR) n'appelle jamais la méthode Finalize sur une structure ; les classes sont arrêtées par le « garbage collector », qui appelle **Finalize** sur une classe lorsqu'il ne détecte aucune référence active restante ;
- une structure n'exige pas de constructeur, contrairement à une classe ;

- les structures peuvent avoir des constructeurs non partagés uniquement lorsqu'ils acceptent des paramètres, les classes peuvent en accepter avec ou sans paramètre.

Chaque structure possède un constructeur public implicite sans paramètres. Ce constructeur initialise toutes les éléments de données de la structure en leur attribuant leurs valeurs par défaut. Vous ne pouvez pas modifier ce comportement.

Instances et variables

Puisque les structures sont des types valeur, chaque variable de structure est définitivement liée à une instance de structure individuelle. En revanche, les classes sont des types référence, et une variable d'objet peut faire référence à plusieurs instances de classe à différents moments. Cette distinction affecte votre utilisation de structures et de classes de diverses façons :

- **Initialisation.** Une variable de structure comprend implicitement une initialisation des éléments à l'aide du constructeur sans paramètre de la structure. En conséquence, `Dim s As struct1` équivaut à `Dim s As struct1 = New struct1()`.

- **Assignation de variables.** Lorsque vous assignez une variable de structure à une autre ou que vous passez une instance de structure à un argument de procédure, les valeurs actuelles de tous les éléments de variable sont copiés à la nouvelle structure. Lorsque vous assignez une variable d'objet à une autre ou que vous passez une variable d'objet à une procédure, seul le pointeur de référence est copié.

- **Assignation de Nothing.** Vous pouvez assigner la valeur Nothing (Visual Basic) à une variable de structure, mais l'instance continue à être associée à la variable. Vous pouvez toujours appeler ses méthodes et accéder à ses éléments de données, même si les éléments de variable sont réinitialisés par l'assignation.

  En revanche, si vous attribuez la valeur **Nothing** à une variable d'objet, vous la dissociez d'une instance de classe et vous ne pouvez pas accéder aux membres par l'intermédiaire de la variable jusqu'à ce que vous lui assigniez une autre instance.

- **Instances multiples.** Une variable objet peut avoir diverses instances de classe qui lui sont assignées à des moments différents et plusieurs variables objet peuvent faire référence simultanément à la même instance de classe. Les modifications apportées aux valeurs des membres de classe affectent ces membres lorsque vous y accédez par l'intermédiaire d'une autre variable pointant vers la même instance.

  Cependant, les éléments de structure sont isolés dans leur propre instance. Les modifications apportées à leurs valeurs ne sont pas réfléchies dans d'autres variables de structure, même dans d'autres instances de la même déclaration **Structure**.

- **Égalité.** Un test d'égalité de deux structures doit être effectué sur un élément après l'autre. Deux variables objet peuvent être comparées à l'aide de la méthode Equals. **Equals** indique si les deux variables pointent vers la même instance.

Voir aussi

Tâches

Dépannage des types de données

Concepts

Types de données en Visual Basic
Types de données composites
Structures et autres éléments de programmation
Types valeur et types référence

Autres ressources

Implémentation des types de données
Structures constituées à partir de vos propres types de données
Présentation des classes

Programmation sans type dans Visual Basic

Le compilateur Visual Basic peut appliquer deux règles qui affectent votre code source :

- Chaque variable locale doit apparaître dans une instruction de déclaration avant d'être utilisée dans tout autre type d'instruction. Cette règle est appliquée par défaut.
- Chaque conversion de type restrictive doit être spécifiée à l'aide d'un mot clé de conversion de type. Cette règle n'est pas appliquée par défaut.

Options du compilateur qui modifient les règles

Vous pouvez changer l'une ou l'autre règle, voire les deux, en réinitialisant les options correspondantes du compilateur. Si vous assouplissez la règle de conversion de type, le compilateur opère dans le cadre d'une *sémantique de type permissive* qui vous permet de pratiquer des conversions restrictives sans mots clés de conversion. Pour plus d'informations, consultez Vérification du type dans Visual Basic.

Si vous assouplissez la règle de déclaration de variable, vous pouvez ensuite faire appel à la *programmation sans type* qui vous permet de référencer des variables que vous n'avez pas déclarées au préalable. Pour plus d'informations, consultez Déclaration implicite et explicite.

La programmation sans type n'est en général pas recommandée parce qu'elle peut entraîner des petites erreurs d'exécution concernant les types des données. Toutefois, vous la trouverez peut-être utile dans certaines circonstances ; par exemple, pour porter une ancienne application ne contenant pas de déclaration de variables.

Voir aussi

Tâches

Dépannage des types de données

Référence

Option Explicit, instruction (Visual Basic)
Option Strict, instruction

Concepts

Types de données en Visual Basic
Caractères de type
Types de données composites
Object comme type de données universel
Utilisation efficace des types de données

Autres ressources

Types de données élémentaires
Conversions de type en Visual Basic

Déclaration implicite et explicite

Par défaut, le compilateur Visual Basic impose la *déclaration explicite*, qui exige que vous déclariez chaque variable avant de l'utiliser. Vous pouvez lever cette exigence et permettre la *déclaration implicite*.

Visual Basic met à votre disposition un commutateur de déclaration explicite. Par défaut, ce commutateur a la valeur **On** et le compilateur met en vigueur la déclaration explicite. Si vous définissez ce commutateur à **Off**, vous pouvez utiliser des variables sans les déclarer.

Commutateur de déclaration explicite. Pour définir à **On** ou **Off** le commutateur de déclaration explicite, vous disposez des méthodes suivantes :

- Définissez la propriété de projet adéquate dans l'environnement de développement intégré (IDE). Sélectionnez **Propriétés de <Nom_Projet>** dans le menu **Projet**, puis sélectionnez l'onglet **Compiler**. Vous pouvez définir les valeurs par défaut pour **Option Explicit**, **Option Strict** et **Option Compare.**
- Spécifiez l'option du compilateur de ligne de commande /optionexplicit.
- Incluez l'instruction Option Explicit (Visual Basic) au début de votre code.

Si vous utilisez l'instruction **Option Explicit**, elle substitue la propriété de projet et les paramètres d'option du compilateur, mais seulement pour le fichier de code source dans lequel elle apparaît.

Avantage de performance. La définition de **Option Explicit** à **On** présente l'avantage de forcer la réalisation de l'inférence du type en phase de compilation plutôt qu'en phase d'exécution. Les performances sont alors améliorées.

Déclaration implicite

Si vous définissez **Option Explicit** à **Off**, vous pouvez déclarer implicitement une variable simplement en l'utilisant dans votre code. Le compilateur assigne le type de données Object à toutes les variables déclarées implicitement. Toutefois, votre application est plus efficace si vous déclarez explicitement toutes vos variables avec un type de données précis. Vous réduisez ainsi les risques d'erreurs de conflit de nom et de fautes de frappe. Vous permettez également au compilateur de détecter des erreurs d'exécution potentielles, telles que l'assignation d'un type **Integer** à un type **Short**.

Erreurs potentielles
Nouvelles variables involontaires

Vous pouvez écrire une procédure dans laquelle vous ne déclarez pas de variable locale. L'exemple suivant illustre ce comportement :

```
Function safeSqrt(num)
' Make sure num is positive for square root.
    tempVal = Math.Abs(num)
    Return Math.Sqrt(tempVal)
End Function
```

Visual Basic crée automatiquement `tempVal` en tant que variable locale que vous pouvez utiliser comme si vous l'aviez explicitement déclarée. Bien que pratique, cette méthode peut entraîner de petites erreurs dans votre code si vous avez mal orthographié un nom de variable. Supposons que vous avez écrit la procédure de l'exemple précédent de la manière suivante :

```
Function safeSqrt(num)
' Make sure num is positive for square root.
    tempVal = Math.Abs(num)
    Return Math.Sqrt(temVal)
End Function
```

À première vue, votre code est le même. Toutefois, comme la variable `tempVal` est mal orthographiée en tant qu'argument de `Sqrt`, le compilateur crée une variable locale supplémentaire appelée `temVal` à laquelle n'est jamais assignée de valeur et votre fonction retourne toujours zéro.

Référence d'élément existant involontaire

Lorsque Visual Basic rencontre un nouveau nom, il ne peut pas déterminer si vous avez souhaité déclarer une nouvelle variable implicitement ou si vous avez mal orthographié un nom de variable existant. Par conséquent, il essaie de créer une nouvelle

variable avec ce nom. Il se peut qu'il y ait une variable ou un autre élément de programmation déjà défini avec ce nom ; dans ce cas, votre code utiliserait cette définition involontairement.

Vous pouvez éviter les problèmes de variables mal orthographiées en utilisant la déclaration explicite.

Déclaration explicite

Si la déclaration explicite avait été active pour le fichier source contenant la procédure `safeSqrt` de l'exemple précédent, Visual Basic aurait reconnu `tempVal` et `temVal` comme des variables non déclarées et aurait généré un message d'erreur pour l'une et l'autre. En conséquence de quoi, vous auriez explicitement déclaré `tempVal`. L'exemple suivant illustre ce comportement :

```
Function safeSqrt(ByVal num As Double) As Double
' Make sure num is positive for square root.
    Dim tempVal As Double = Math.Abs(num)
    Return Math.Sqrt(temVal)
End Function
```

Avec ce nouveau code, vous comprendriez immédiatement le problème, car Visual Basic afficherait un message d'erreur pour la variable `temVal` mal orthographiée. Comme la déclaration explicite vous aide à intercepter les erreurs de ce type, il est recommandé de l'utiliser dans tout votre code.

Remarque

L'instruction **Option Explicit** opère fichier par fichier. Elle doit figurer au début de chaque fichier de code source dans lequel vous voulez pouvoir contrôler la déclaration explicite de variable.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Référence

Liste des types de données (Visual Basic)

Fonctions de conversion de types de données

Concepts

Programmation sans type dans Visual Basic

Vérification du type dans Visual Basic

Object comme type de données universel

Utilisation efficace des types de données

Variables en Visual Basic

Conventions d'affectation de noms Visual Basic

Vérification du type dans Visual Basic

Pour les conversions de types de données, le compilateur Visual Basic peut fonctionner avec une sémantique de type stricte ou permissive. Si la *sémantique de type stricte* est active, seules les conversions étendues sont implicitement autorisées et les conversions restrictives doivent être explicites. Si la *sémantique de type permissive* est active, vous pouvez tenter d'effectuer toutes les conversions étendues et restrictives de manière implicite. Ces sémantiques s'appliquent aux conversions entre tous les types de données, y compris les types d'objets.

Méthodes pour définir l'option de vérification de type

Visual Basic met à votre disposition un commutateur de contrôle de vérification de type. Par défaut, ce commutateur est **Off** et le compilateur autorise les conversions restrictives implicites. Si vous définissez le commutateur à **On**, le compilateur met en vigueur la sémantique de type stricte.

Commutateur de vérification de type

Pour affecter la valeur **On** ou **Off** à ce commutateur, vous disposez des méthodes suivantes :

- Définition de la propriété de projet appropriée dans la fenêtre **Propriétés** de l'environnement de développement intégré (IDE, Integrated Development Environment)
- Spécification de l'option du compilateur de ligne de commande /optionstrict
- Ajout de l'instruction **Option Strict** au début de votre code

Si vous utilisez l'instruction **Option Strict**, elle substitue la propriété de projet et les paramètres d'option du compilateur, mais seulement pour le fichier de code source dans lequel elle apparaît. Pour plus d'informations, consultez Option Strict, instruction.

Avantage de performances

La définition de **Option Strict** à **On** présente l'avantage de forcer une liaison anticipée. Les performances sont alors améliorées.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données

Concepts

Types de données en Visual Basic
Programmation sans type dans Visual Basic
Déclaration implicite et explicite
Object comme type de données universel
Utilisation efficace des types de données
Conversions étendues et restrictives
Conversions implicites et explicites

Object comme type de données universel

Le type de données Object est le type racine dans le .NET Framework et dans Visual Basic. Cela signifie que tous les autres types de données et types d'objets sont dérivés de lui, directement ou non. Cela signifie également que tout autre type de données, élémentaire ou composite, peut être converti vers le type **Object**.

Typage faible

Vous pouvez utiliser **Object** comme type de données universel. C'est ce qu'on appelle le *typage faible*. L'exemple suivant illustre la souplesse du typage faible.

```
Dim v As Object
v = "17"
' v contains the 2-character String value "17".
v = v - 15
' v now contains the Integer value 2.
v = "H" & v
' v now contains the 2-character String value "H2".
```

Si vous pouvez effectuer de telles opérations sur des variables **Object** sans trop vous soucier du type des données qu'elles contiennent, il y a malgré tout des pièges à éviter :

- Si vous effectuez des opérations ou des fonctions arithmétiques sur un **Object**, il doit contenir des données numériques sinon une erreur d'exécution se produit.
- Si vous concaténez des chaînes, utilisez l'opérateur **&** à la place de l'opérateur **+**. Il y a des règles complexes qui déterminent quand l'opérateur **+** ajoute ses opérandes et quand il les concatène. L'opérateur **+** doit effectuer également une vérification et une conversion de type dans certains cas, ce qui réduit les performances.

Modification du type de données d'une variable objet

Bien qu'une variable **Object** accepte des données de tous types, vous ne pouvez pas modifier la variable elle-même en un type de données différent. Toutefois, vous pouvez redimensionner une variable **Object** en un tableau de variables **Object**. L'exemple suivant illustre l'utilisation valide et non valide de l'ReDim, instruction (Visual Basic).

```
' The following statement declares a single Object.
Dim someObj As Object
' The following reallocation is valid only for Object.
ReDim someObj(8)
' The following statement attempts an INVALID change of data type.
ReDim someObj(8) As Double
```

La première instruction **ReDim** change `someObj` en un tableau de type **Object**. Cette opération n'est correcte qu'avec le type de données **Object**. La seconde instruction **ReDim** est non valide, parce qu'elle fait intervenir un autre type de données. Vous pouvez obtenir l'effet souhaité en utilisant un tableau séparé. L'exemple suivant illustre ce comportement :

```
' First allocate a separate array.
Dim someArray(8) As Double
' Then assign the new array to the Object variable.
someObj = someArray
```

Object comme alternative à une structure

Étant donné que le type de données **Object** est capable d'enregistrer beaucoup de types de données différents, vous pouvez, dans de nombreux cas, utiliser un tableau **Object** à la place d'une structure. Un tableau **Object** offre un peu plus de souplesse qu'une structure, parce que vous pouvez à tout moment modifier le type des données enregistrées dans chaque élément et lui conférer un caractère dynamique afin de pouvoir le redimensionner à votre gré. Toutefois, un tableau **Object** utilise davantage de mémoire que la structure équivalente et ses performances sont ralenties.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données

Concepts

Programmation sans type dans Visual Basic
Déclaration implicite et explicite
Vérification du type dans Visual Basic
Utilisation efficace des types de données
Déclaration des variables objets
Assignation des variables objets
Valeurs des variables objets

Autres ressources

Conversions de type en Visual Basic

Utilisation efficace des types de données

Les variables non déclarées et celles qui sont déclarées sans type de données prennent le type de données **Object**. Vous pouvez ainsi écrire plus facilement et rapidement des programmes, mais leur exécution sera plus lente.

Typage fort

Le processus qui consiste à spécifier des types de données pour toutes les variables est appelé *typage fort*. Le typage fort présente plusieurs avantages :

- Permet la prise en charge par IntelliSense® de vos variables. Il vous permet de voir leurs propriétés et leurs autres membres à mesure que vous tapez le code.
- Tire parti du contrôle de type de compilateur. Il intercepte les instructions susceptibles de provoquer, au moment de l'exécution, des erreurs telles qu'un dépassement de capacité. Il intercepte également les appels de méthodes dans des objets qui ne les prennent pas en charge.
- Il permet ainsi à votre code de s'exécuter plus rapidement.

Types de données les plus efficaces

Pour les variables qui ne contiennent jamais de fractions, les types de données intégraux sont plus efficaces que les types non intégraux. Dans Visual Basic, **Integer** et **UInteger** sont les types numériques les plus efficaces.

Pour les nombres fractionnaires, **Double** est le type de données le plus efficace, parce que les processeurs sur les plates-formes actuelles exécutent des opérations en virgule flottante double précision. Toutefois, les opérations avec **Double** ne sont pas aussi rapides qu'avec les types intégraux, tels que **Integer**.

Spécifier le type de données

Faites appel à Dim, instruction (Visual Basic) pour déclarer une variable d'un type spécifique. Vous pouvez spécifier simultanément son niveau d'accès à l'aide du mot clé Public (Visual Basic), Protected (Visual Basic), Friend (Visual Basic) ou Private (Visual Basic), comme dans l'exemple suivant.

```
Private x As Double
Protected s As String
```

Conversion des caractères

Les fonctions **AscW** et **ChrW** opèrent en Unicode. Utilisez-les de préférence à **Asc** et **Chr** qui doivent traduire en Unicode ou à partir de ce format.

Voir aussi

Référence

Fonctions Asc, AscW
Fonctions Chr, ChrW

Concepts

Types de données en Visual Basic
Programmation sans type dans Visual Basic
Déclaration implicite et explicite
Vérification du type dans Visual Basic
Object comme type de données universel
Types de données numériques
Déclaration de variable dans Visual Basic

Autres ressources

Utilisation de la fonctionnalité IntelliSense

Dépannage des types de données

Cette page répertorie des problèmes courants qui peuvent se produire lorsque vous exécutez des opérations sur des types de données intrinsèques.

Les expressions à virgule flottante ne sont pas considérées comme égales

Lorsque vous utilisez des nombres à virgule flottante (Single, type de données (Visual Basic) et Double, type de données (Visual Basic)), n'oubliez pas qu'ils sont stockés sous forme de fractions binaires. En d'autres termes, ils ne peuvent pas contenir une représentation exacte d'une quantité qui n'est pas une fraction binaire (sous la forme *k / (2 ^ n)* où *k* et *n* désignent des entiers). Par exemple, 0,5 (= 1/2) et 0,3125 (= 5/16) peuvent être considérés comme des valeurs précises, tandis que 0,2 (= 1/5) et 0,3 (= 3/10) peuvent être uniquement des approximations.

En raison de cette imprécision, vous ne pouvez pas vous baser sur des résultats exacts lorsque vous utilisez des valeurs à virgule flottante. En particulier, deux valeurs qui sont théoriquement égales peuvent présenter des représentations légèrement différentes.

Pour comparer des quantités à virgule flottante

1. Calculez la valeur absolue de leur différence, à l'aide de la méthode Abs de la classe Math dans l'espace de noms System.
2. Déterminez une différence maximale acceptable : par exemple, vous pouvez considérer les deux quantités comme étant égales pour des raisons pratiques si leur différence n'est pas supérieure.
3. Comparez la valeur absolue de la différence à la différence acceptable.

L'exemple suivant présente une comparaison correcte et incorrecte de deux valeurs **Double**.

VB

```
Dim oneThird As Double = 1.0 / 3.0
Dim pointThrees As Double = 0.333333333333333

' The following comparison does not indicate equality.
Dim exactlyEqual As Boolean = (oneThird = pointThrees)

' The following comparison indicates equality.
Dim closeEnough As Double = 0.000000000000001
Dim absoluteDifference As Double = Math.Abs(oneThird - pointThrees)
Dim practicallyEqual As Boolean = (absoluteDifference < closeEnough)

MsgBox("1.0 / 3.0 is represented as " & oneThird.ToString("G17") _
  & vbCrLf & "0.333333333333333 is represented as " _
  & pointThrees.ToString("G17") _
  & vbCrLf & "Exact comparison generates " & CStr(exactlyEqual) _
  & vbCrLf & "Acceptable difference comparison generates " _
  & CStr(practicallyEqual))
```

L'exemple précédent utilise la méthode ToString de la structure Double pour spécifier une plus grande précision que le mot clé **CStr**. La valeur par défaut est 15 chiffres, mais le format « G17 » l'étend à 17 chiffres.

L'opérateur Mod ne retourne pas de résultat précis

En raison de l'imprécision du stockage à virgule flottante, Mod, opérateur (Visual Basic) peut retourner un résultat inattendu lorsqu'au moins un des opérandes est à virgule flottante.

Decimal, type de données (Visual Basic) n'utilise pas la représentation à virgule flottante. Plusieurs nombres qui sont inexacts dans **Single** et **Double** sont exacts dans **Decimal** (par exemple, 0,2 et 0,3). Bien que l'arithmétique soit plus lente en **Decimal** qu'en virgule flottante, cela peut valoir la peine de réduire les performances pour obtenir une plus grande précision.

Pour rechercher le reste entier de quantités à virgule flottante

1. Déclarez les variables comme **Decimal**.
2. Utilisez le caractère de type de littéral **D** pour forcer les littéraux sur la valeur **Decimal**, au cas où leurs valeurs sont trop

grandes pour le type de données **Long**.

L'exemple suivant montre l'imprécision potentielle des opérandes à virgule flottante.

VB

```
Dim two As Double = 2.0
Dim zeroPointTwo As Double = 0.2
Dim quotient As Double = two / zeroPointTwo
Dim doubleRemainder As Double = two Mod zeroPointTwo

MsgBox("2.0 is represented as " & two.ToString("G17") _
  & vbCrLf & "0.2 is represented as " & zeroPointTwo.ToString("G17") _
  & vbCrLf & "2.0 / 0.2 generates " & quotient.ToString("G17") _
  & vbCrLf & "2.0 Mod 0.2 generates " _
  & doubleRemainder.ToString("G17"))

Dim decimalRemainder As Decimal = 2D Mod 0.2D
MsgBox("2.0D Mod 0.2D generates " & CStr(decimalRemainder))
```

L'exemple précédent utilise la méthode **ToString** de la structure **Double** pour spécifier une plus grande précision que le mot clé **CStr**. La valeur par défaut est 15 chiffres, mais le format « G17 » l'étend à 17 chiffres.

Étant donné que `zeroPointTwo` a la valeur **Double**, sa valeur pour 0,2 est une fraction binaire se répétant à l'infini avec une valeur stockée de 0,20000000000000001. Lorsqu'on divise 2,0 par cette quantité, on obtient 9,9999999999999995 avec un reste de 0,19999999999999991.

Dans l'expression pour `decimalRemainder`, le caractère de type de littéral `D` force les deux opérandes sur la valeur **Decimal** et 0,2 a une représentation précise. Par conséquent, l'opérateur **Mod** obtient le reste attendu de 0,0.

Notez que cela n'est pas suffisant pour déclarer `decimalRemainder` comme **Decimal**. Vous devez également forcer les littéraux sur la valeur **Decimal**, ou ils prennent la valeur par défaut **Double** et `decimalRemainder` reçoit la même valeur inexacte que `doubleRemainder`.

Impossible de convertir correctement le type Boolean en type numérique

Les valeurs Boolean, type de données (Visual Basic) ne sont pas stockées en tant que nombres et les valeurs stockées ne sont pas destinées à être équivalentes aux nombres. Pour garantir la compatibilité avec les versions antérieures, Visual Basic fournit des mots clés de conversion (CType Function, **CBool**, **CInt**, etc.) pour effectuer des conversions entre des types **Boolean** et numériques. Toutefois, d'autres langages exécutent parfois différemment ces conversions, comme c'est le cas pour les méthodes du .NET Framework.

Vous ne devez jamais écrire un code qui repose sur les valeurs numériques équivalentes à **True** et **False**. Autant que possible, limitez l'utilisation des variables **Boolean** aux valeurs logiques pour lesquelles elles sont conçues. S'il s'avère nécessaire de mélanger des valeurs **Boolean** et des valeurs numériques, assurez-vous que vous comprenez la méthode de conversion choisie.

Conversion dans Visual Basic

Lorsque vous utilisez les mots clés de conversion **CType** ou **CBool** pour convertir des types de données numériques en **Boolean**, 0 devient **False** et toutes les autres valeurs deviennent **True**. Lorsque vous convertissez des valeurs **Boolean** en types numériques à l'aide de mots clés de conversion, **False** devient 0 et **True** devient -1.

Conversion dans la structure

La méthode ToInt32 de la classe Convert dans l'espace de noms **System** convertit **True** en +1.

Si vous devez convertir une valeur **Boolean** en un type de données numériques, choisissez attentivement la méthode de conversion à utiliser.

Un littéral de caractère génère une erreur de compilation

En l'absence de caractères de type, Visual Basic utilise des types de données par défaut pour les littéraux. Le type par défaut pour un littéral de caractère — mis entre guillemets doubles (**" "**) — est **String**.

Le type de données **String** ne s'étend pas au Char, type de données (Visual Basic). Cela signifie que si vous souhaitez assigner un littéral à une variable **Char**, vous devez soit effectuer une conversion restrictive soit forcer le littéral sur le type **Char**.

Pour créer un littéral de caractère à assigner à une variable ou une constante

1. Déclarez la variable ou la constante comme **Char**.
2. Entourez la valeur du caractère par des guillemets doubles (**" "**).
3. Faites suivre les guillemets doubles fermants du caractère de type de littéral **C** pour forcer le littéral sur la valeur **Char**. Cette procédure est nécessaire si le commutateur de vérification de type (instruction Option Strict) a la valeur **On**, et elle est recommandée dans tous les cas.

L'exemple suivant présente des assignations infructueuses et réussies d'un littéral sur une variable **Char**.

VB

```
Option Strict On
```

VB

```
Dim charVar As Char
' The following statement attempts to convert a String literal to Char.
' Because Option Strict is On, it generates a compiler error.
charVar = "Z"
' The following statement succeeds because it specifies a Char literal.
charVar = "Z"c
' The following statement succeeds because it converts String to Char.
charVar = CChar("Z")
```

Il existe toujours un risque lié à l'utilisation de conversions restrictives, car elles peuvent échouer au moment de l'exécution. Par exemple, une conversion de **String** en **Char** peut échouer si la valeur **String** contient plusieurs caractères. Par conséquent, l'utilisation du caractère de type **C** est préconisée.

Échec de la conversion de chaîne au moment de l'exécution

String, type de données (Visual Basic) participe à très peu de conversions étendues. **String** s'étend uniquement à lui-même et à **Object**, et seuls **Char** et **Char()** (tableau **Char**) s'étendent à **String**. Cela s'explique par le fait que les variables et les constantes **String** peuvent contenir des valeurs qui ne sont pas prises en charge par d'autres types de données.

Lorsque le commutateur de vérification de type (instruction Option Strict) a la valeur **On**, le compilateur rejette toutes les conversions restrictives implicites, y compris celles qui impliquent **String**. Votre code peut toujours utiliser les mots clés de conversion tels que **CStr** et CType Function, qui invitent le .NET Framework à tenter la conversion.

Protection de la conversion restrictive

L'inconvénient des conversions restrictives est qu'elles peuvent échouer au moment de l'exécution. Par exemple, si une variable **String** contient une valeur autre que « True » ou « False », elle ne peut pas être convertie en **Boolean**. Si elle contient des caractères de ponctuation, la conversion en un type numérique échoue. Vous ne devez pas tenter de conversion sauf si vous savez que votre variable **String** contient toujours des valeurs que le type de destination peut accepter.

Si vous devez effectuer une conversion de **String** en un autre type de données, la procédure la plus sûre consiste à insérer la conversion tentée dans Try...Catch...Finally, instruction (Visual Basic). Cela vous permet de gérer les défaillances au moment de l'exécution.

Tableaux de caractères

Un élément **Char** unique et un tableau d'éléments **Char** s'étendent tous deux à **String**. Toutefois, **String** ne s'étend pas à **Char()**. Pour convertir une valeur **String** en un tableau **Char**, vous pouvez utiliser la méthode ToCharArray de la classe System.String.

Valeurs sans signification

En général, les valeurs **String** ne sont pas explicites dans d'autres types de données, et la conversion est très artificielle et dangereuse. Autant que possible, limitez l'utilisation des variables **String** aux séquences de caractère pour lesquelles elles sont conçues. Vous ne devez jamais écrire un code qui dépend de valeurs équivalentes dans d'autres types.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Fonctions de conversion de types de données

Concepts

Types de données en Visual Basic
Caractères de type
Programmation sans type dans Visual Basic
Les types de données sont des types .NET Framework
Utilisation efficace des types de données

Autres ressources

Implémentation des types de données
Conversions de type en Visual Basic

Variables en Visual Basic

Il est parfois nécessaire de stocker des valeurs lors de l'exécution de calculs avec Visual Basic. Par exemple, vous pouvez être amené à calculer plusieurs valeurs, à les comparer et à exécuter différentes opérations sur ces valeurs, en fonction du résultat de la comparaison. Vous devez conserver les valeurs si vous souhaitez les comparer.

Utilisation

Visual Basic, comme la plupart des langages de programmation, utilise des variables pour le stockage des valeurs. Une *variable* possède un nom (le mot utilisé pour faire référence à la valeur contenue dans la variable) et un type de données (qui détermine la nature des données pouvant être stockées dans la variable). Une variable peut être représentée par un tableau si elle doit stocker un ensemble indexé d'éléments de données étroitement liés.

Assignation de valeurs

Vous devez utiliser les instructions d'assignation pour effectuer des calculs et assigner le résultat à une variable, comme indiqué dans l'exemple suivant :

VB

```
' The following statement assigns the value 10 to the variable.
applesSold = 10
' The following statement increments the variable.
applesSold = applesSold + 1
' The variable now holds the value 11.
```

Remarque

Dans cet exemple, le signe égal (=) représente un opérateur d'assignation, et non un opérateur d'égalité. La valeur est assignée à la variable `applesSold`.

Pour plus d'informations, consultez Comment : placer des données dans et en dehors d'une variable.

Variables et propriétés

À l'instar d'une variable, une *propriété* représente une valeur à laquelle vous pouvez accéder. Toutefois, elle est plus complexe qu'une variable. Une propriété utilise des blocs de code qui contrôlent la définition et la récupération de sa valeur. Pour plus d'informations, consultez Différences entre les propriétés et les variables en Visual Basic.

Voir aussi

Tâches

Dépannage des variables en Visual Basic
Comment : placer des données dans et en dehors d'une variable

Concepts

Déclaration de variable dans Visual Basic
Variables objet dans Visual Basic
Différences entre les propriétés et les variables en Visual Basic

Déclaration de variable dans Visual Basic

Vous devez déclarer une variable pour spécifier son nom et ses caractéristiques. L'instruction de déclaration pour les variables est l'instruction Dim (Visual Basic). Son emplacement et son contenu déterminent les caractéristiques de la variable.

Pour plus d'informations sur les règles et les considérations relatives à l'affectation de noms de variables, consultez Noms d'éléments déclarés.

Niveaux de déclaration

Variables locales et membres

Une *variable locale* est une variable déclarée dans une procédure. Une *variable membre* est un membre d'un type Visual Basic ; elle est déclarée au niveau module, à l'intérieur d'une classe, d'une structure ou d'un module, mais pas dans une procédure interne à cette classe, structure ou module.

Variables d'instance et variables partagées

Dans une classe ou une structure, la catégorie d'une variable membre dépend du fait qu'elle est partagée ou non. Si elle est déclarée à l'aide du mot clé Shared, il s'agit d'une *variable partagée* dont il n'existe qu'une seule copie partagée entre toutes les instances de la classe ou de la structure.

Sinon, il s'agit d'une *variable d'instance* dont la copie séparée est créée pour chaque instance de la classe ou de la structure. Une copie donnée d'une variable d'instance n'est disponible que pour l'instance pour laquelle elle a été créée. Elle est indépendante d'une copie dans toute autre instance.

Déclaration du type de données

La clause As dans l'instruction de déclaration vous permet de définir le type de données ou le type d'objet de la variable déclarée. Vous pouvez spécifier l'un des types suivants pour une variable :

- Type de données élémentaire, tel que **Boolean**, **Long** ou **Decimal**
- Type de données composite, tel qu'un tableau ou une structure
- Type d'objet, ou classe, défini dans votre application ou dans une autre application
- Classe du .NET Framework, telle que Label ou TextBox
- Type d'interface, tel que IComparable ou IDisposable

Vous pouvez déclarer plusieurs variables dans une instruction sans devoir répéter le type de données. Dans les instructions suivantes, les variables `i`, `j` et `k` sont déclarées comme type **Integer**, `l` et `m` comme **Long** et `x` et `y` comme **Single** :

```
Dim i, j, k As Integer
' All three variables in the preceding statement are declared as Integer.
Dim l, m As Long, x, y As Single
' In the preceding statement, l and m are Long, x and y are Single.
```

Pour plus d'informations sur les types de données, consultez Types de données en Visual Basic. Pour plus d'informations sur les objets, consultez Programmation orientée objet dans Visual Basic et Programmation à l'aide de composants.

Déclaration des caractéristiques

La *durée de vie* d'une variable correspond à la durée pendant laquelle elle peut être utilisée. En général, une variable existe tant que l'élément qui la déclare (tel qu'une procédure ou une classe) continue à exister. Dans certains cas, il est possible d'étendre la durée de vie d'une variable. Pour plus d'informations, consultez Durée de vie dans Visual Basic.

La *portée* d'une variable correspond à l'ensemble du code qui peut s'y référer sans qualifier son nom. La portée d'une variable est déterminée par l'emplacement où elle est déclarée. Le code situé dans une zone donnée peut utiliser les variables définies dans cette zone sans devoir qualifier leurs noms. Pour plus d'informations, consultez Portée dans Visual Basic.

Le *niveau d'accès* d'une variable est l'étendue de code qui a l'autorisation d'y accéder. Il est déterminé par le modificateur d'accès (tel que Public (Visual Basic) ou Private (Visual Basic)) que vous utilisez dans l'instruction **Dim**. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Voir aussi

Tâches

Comment : créer une variable

Comment : créer une variable qui ne change pas de valeur

Comment : placer des données dans et en dehors d'une variable

Référence

Liste des types de données (Visual Basic)

Protected (Visual Basic)

Friend (Visual Basic)

Static (Visual Basic)

Concepts

Choix du type de variable à définir

Caractéristiques d'éléments déclarés

Comment : créer une variable

Créez une variable à l'aide d'un Dim, instruction (Visual Basic).

Pour créer une variable

1. Déclarez la variable dans une instruction **Dim**.

```
Dim newCustomer
```

2. Insérez les spécifications pour les caractéristiques de la variable, telles que Private (Visual Basic), Static (Visual Basic), Shadows ou WithEvents. Pour plus d'informations, consultez Caractéristiques d'éléments déclarés.

```
Public Static newCustomer
```

   Vous n'avez pas besoin du mot clé **Dim** si vous utilisez d'autres mots clés dans la déclaration.

3. Faites suivre les spécifications du nom de la variable qui doit se conformer aux règles et aux conventions Visual Basic. Pour plus d'informations, consultez Noms d'éléments déclarés.

```
Public Static newCustomer
```

4. Faites suivre le nom de la clause As pour spécifier le type de données de la variable.

```
Public Static newCustomer As Customer
```

   Si vous ne spécifiez pas le type de données, la valeur par défaut est **Object**.

5. Faites suivre la clause **As** d'un signe égal (**=**) et faites suivre le signe égal de la valeur initiale de la variable.

   Visual Basic assigne la valeur spécifiée à la variable à chaque exécution de l'instruction **Dim**. Si vous ne spécifiez pas de valeur initiale, Visual Basic assigne la valeur initiale par défaut pour le type de données de la variable lorsqu'elle entre pour la première fois dans le code contenant l'instruction **Dim**.

   Si la variable est un type référence, vous pouvez créer une instance de sa classe en insérant le mot clé New (Visual Basic) dans la clause **As**. Si vous n'utilisez pas **New**, la valeur initiale de la variable est Nothing (Visual Basic).

```
Public Static newCustomer As New Customer
```

Voir aussi

Concepts

Variables en Visual Basic
Déclaration de variable dans Visual Basic
Noms d'éléments déclarés
Caractéristiques d'éléments déclarés
Types valeur et types référence
Instructions de déclaration en Visual Basic

Choix du type de variable à définir

Lorsque vous définissez une variable, vous devez décider des caractéristiques suivantes :

- Son *type de données* - quel genre de données elle doit contenir
- Sa *durée de vie* - combien de temps elle doit continuer à exister
- Sa *portée* - quel code doit être capable d'y faire référence sans qualifier son nom
- Son *niveau d'accès* - quel code doit avoir l'autorisation en lecture et en écriture sur elle

Type de données

Dans le Dim, instruction (Visual Basic) qui déclare la variable, insérez une clause As spécifiant le type de données approprié (tel que **Integer** ou **String**). Les pages suivantes peuvent vous aider à choisir le type de données d'une variable.

- Comment : stocker des entiers dans une variable
- Comment : stocker le plus grand nombre possible dans une variable
- Comment : optimiser le stockage d'entiers positifs avec les types non signés
- Comment : stocker des fractions dans une variable
- Comment : stocker les chiffres les plus significatifs dans une variable
- Comment : stocker des valeurs monétaires dans une variable
- Comment : stocker des caractères dans une variable
- Comment : stocker des valeurs True et False dans une variable
- Comment : stocker des valeurs de date et d'heure dans une variable
- Comment : stocker des données de type inconnu dans une variable
- Comment : déclarer une structure
- Création et utilisation d'objets

Pour plus d'informations, consultez Liste des types de données (Visual Basic).

Durée de vie

La décision importante concernant la durée de vie est de déterminer s'il est acceptable ou non pour la variable de cesser d'exister lorsque le module, la classe ou la procédure qui la déclare cesse d'exister.

S'il n'est pas nécessaire que la variable excède la durée de vie de l'élément qui la contient, aucune action n'est nécessaire. Si la variable doit continuer d'exister plus longtemps que l'élément qui la contient, vous pouvez inclure les mots clés **Static** ou **Shared** dans son instruction **Dim**. Suivez les étapes décrites dans Comment : augmenter la durée de vie d'une variable.

Pour plus d'informations, consultez Durée de vie dans Visual Basic.

Portée

La portée d'une variable est normalement identique à son espace de déclaration, c'est-à-dire, l'élément conteneur dans lequel elle est déclarée. Vous devez décider le degré d'extension de la portée de la variable.

Vérifiez que l'instruction **Dim** apparaît au niveau approprié, comme le niveau du bloc, de la procédure ou du module. Suivez les étapes décrites dans Comment : contrôler la portée d'une variable.

Pour plus d'informations, consultez Portée dans Visual Basic.

Niveau d'accès

Chaque variable possède un niveau d'accès par défaut qui dépend du lieu où elle est déclarée, c'est-à-dire du type d'élément conteneur.

Si vous devez spécifier un niveau d'accès autre que la valeur par défaut, vous pouvez inclure un modificateur d'accès (tel que

Protected ou **Private**) dans son instruction **Dim**. Cette opération n'est possible que pour les variables membre (à savoir, les variables déclarées à l'extérieur d'une procédure). Suivez les étapes décrites dans Comment : contrôler la disponibilité d'une variable.

Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Voir aussi

Tâches

Comment : augmenter la durée de vie d'une variable
Comment : contrôler la portée d'une variable

Référence

As (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic
Caractéristiques d'éléments déclarés
Portée dans Visual Basic
Niveaux d'accès dans Visual Basic

Comment : créer une variable qui ne change pas de valeur

La notion de variable qui ne change pas de valeur peut sembler contradictoire. Toutefois, il peut arriver qu'une constante ne soit pas possible et il est alors utile d'avoir une variable avec une valeur fixe. Dans une telle situation, vous pouvez définir une variable membre avec le mot clé ReadOnly (Visual Basic).

Vous ne pouvez pas utiliser Const, instruction (Visual Basic) pour déclarer et assigner une valeur de constante dans les circonstances suivantes :

- L'instruction **Const** n'accepte pas le type de données que vous souhaitez utiliser.
- Vous ne connaissez pas la valeur au moment de la compilation.
- Vous êtes incapable de calculer la valeur de constante au moment de la compilation.

Pour créer une variable qui ne change pas de valeur

1. Au niveau du module, déclarez une variable membre avec Dim, instruction (Visual Basic) et incluez le mot clé ReadOnly (Visual Basic).

   ```
   Dim ReadOnly timeStarted
   ```

   Vous pouvez spécifier **ReadOnly** uniquement sur une variable membre. Cela signifie que vous devez définir la variable au niveau du module, en dehors de toute procédure.

2. Si vous pouvez calculer la valeur dans une seule instruction au moment de la compilation, utilisez une clause d'initialisation dans l'instruction **Dim**. Faites suivre la clause As d'un signe égal (**=**), suivi d'une expression. Assurez-vous que le compilateur peut évaluer cette expression à une valeur de constante.

   ```
   Dim ReadOnly timeStarted As Date = Now
   ```

   Vous pouvez assigner une valeur à une variable **ReadOnly** une seule fois. Après cette opération, aucun code ne peut jamais modifier sa valeur.

   Si vous ne connaissez pas la valeur au moment de la compilation ou si vous ne pouvez pas la calculer au moment de la compilation dans une seule instruction, vous pouvez toujours l'assigner au moment de l'exécution dans un constructeur. Pour ce faire, vous devez déclarer la variable **ReadOnly** au niveau de la classe ou de la structure. Dans le constructeur pour cette classe ou cette structure, calculez la valeur fixe de la variable et assignez-la à la variable avant de quitter le constructeur.

Voir aussi

Référence

WriteOnly
Const, instruction (Visual Basic)

Comment : placer des données dans et en dehors d'une variable

Vous pouvez stocker une valeur dans une variable en plaçant le nom de la variable sur le côté gauche d'une instruction d'assignation.

Placement de données dans une variable

Pour stocker une valeur dans une variable

- Spécifiez le nom de variable sur le côté gauche d'une instruction d'assignation.

  L'exemple suivant définit la valeur de la variable `alpha`.

  ```
  alpha = (beta * 6.27) / (gamma + 2.1)
  ```

  La valeur générée du côté droit de l'instruction d'assignation est stockée dans la variable.

Obtention de données à partir d'une variable

Vous pouvez récupérer la valeur d'une variable en incluant le nom de variable dans une expression.

Pour récupérer une valeur à partir d'une variable

- Spécifiez le nom de variable dans une expression. Vous pouvez utiliser une variable aux mêmes emplacements qu'une constante ou un littéral, sauf dans une expression qui définit la valeur d'une constante.

  - ou -

- Spécifiez le nom de variable après le signe égal (**=**) dans une instruction d'assignation.

  L'exemple suivant lit la valeur de la variable `startValue`, puis utilise la valeur de la variable `counter` dans une expression.

  ```
  counter = startValue
  cellValue = (counter + 5) ^ 2
  ```

  La valeur de la variable participe à l'expression comme le ferait une constante, puis elle est stockée dans la variable ou la propriété située sur le côté gauche de l'instruction d'assignation.

Voir aussi

Concepts

Variables en Visual Basic
Déclaration de variable dans Visual Basic
Variables objet dans Visual Basic

Variables objet dans Visual Basic

Outre le stockage direct de valeurs, une variable peut faire référence à un objet. Vous assignez un objet à une variable pour les mêmes raisons que vous assignez une valeur à une variable :

- Un nom de variable est souvent plus court et plus facile à se rappeler que le chemin d'accès complet des méthodes et propriétés permettant d'accéder à l'objet lui-même.
- L'utilisation d'une variable faisant référence à un objet est souvent plus efficace qu'un accès répété à cet objet via les méthodes ou propriétés nécessaires.
- Vous pouvez modifier une variable pour faire référence à d'autres objets durant l'exécution de votre code.

Raccourcissement du code

Vous pouvez utiliser des variables objets pour réduire la longueur du code à taper. L'exemple suivant utilise le chemin d'accès complet de méthodes et de propriétés pour accéder à un objet Control.

```
' Assume Me is a valid Form, or replace Me with a valid Form.
Me.ActiveForm.ActiveControl.Text = "Test"
Me.ActiveForm.ActiveControl.Location = New Point(100, 100)
Me.ActiveForm.ActiveControl.Show()
```

Vous pouvez réduire la longueur de ce code, et accélérer son exécution, si vous utilisez une variable objet en guise de contrôle. Vous devez déclarer la variable objet à l'aide de la classe spécifique que vous comptez lui assigner (`Control` dans le cas présent). Une fois que vous assignez un objet à la variable, vous pouvez traiter celle-ci de la même façon que l'objet auquel elle fait référence. Vous pouvez définir ou récupérer les propriétés de l'objet ou utiliser n'importe laquelle de ses méthodes. L'exemple suivant utilise une variable objet pour simplifier le code de l'exemple précédent.

```
Dim ctrlActv As System.Windows.Forms.Control = Me.ActiveForm.ActiveControl
ctrlActv.Text = "Test"
ctrlActv.Location = New Point(100, 100)
ctrlActv.Show()
```

Voir aussi

Tâches

Comment : accélérer l'accès à un objet comportant un chemin d'accès de qualification long

Concepts

Déclaration de variable dans Visual Basic
Déclaration des variables objets
Assignation des variables objets
Valeurs des variables objets

Comment : accélérer l'accès à un objet comportant un chemin d'accès de qualification long

Si vous accédez fréquemment à un objet qui requiert un chemin d'accès de qualification de plusieurs méthodes et propriétés, vous pouvez accélérer votre code en évitant de répéter le chemin d'accès de qualification.

Vous pouvez éviter de répéter le chemin d'accès de qualification de deux manières. Vous pouvez assigner l'objet à une variable ou vous pouvez l'utiliser dans un bloc **With**...**End With**.

Pour accélérer l'accès à un objet très qualifié en l'assignant à une variable

1. Déclarez une variable du type de l'objet auquel vous accédez fréquemment. Spécifiez le chemin d'accès de qualification dans la partie d'initialisation de la déclaration.

```
Dim ctrlActv As Control = someForm.ActiveForm.ActiveControl
```

2. Utilisez la variable pour accéder aux membres de l'objet.

```
ctrlActv.Text = "Test"
ctrlActv.Location = New Point(100, 100)
ctrlActv.Show()
```

Pour accélérer l'accès à un objet très qualifié à l'aide d'un bloc With...End With

1. Placez le chemin d'accès de qualification dans une instruction **With**.

```
With someForm.ActiveForm.ActiveControl
```

2. Accédez aux membres de l'objet à l'intérieur du bloc **With** avant l'instruction **End With**

```
    .Text = "Test"
    .Location = New Point(100, 100)
    .Show()
End With
```

Voir aussi

Référence

With...End With, instruction (Visual Basic)

Concepts

Variables objet dans Visual Basic

Déclaration des variables objets

Utilisez une instruction de déclaration normale pour déclarer une variable objet. Pour le type de données, spécifiez soit **Object** (c'est-à-dire le type de données Object) soit une classe plus spécifique à partir de laquelle l'objet doit être créé.

Déclarer une variable comme **Object** revient à la déclarer comme System.Object.

Lorsque vous déclarez une variable à l'aide d'une classe d'objet spécifique, elle peut accéder à toutes les méthodes et propriétés exposées par cette classe et les classes dont elle hérite. Si vous déclarez la variable à l'aide de **Object**, elle peut accéder uniquement aux membres de la classe **Object**, sauf si vous tournez **Option Strict Off** pour autoriser la liaison tardive.

Syntaxe de déclaration

Utilisez la syntaxe suivante pour déclarer une variable objet :

```
Dim variablename As [New] { objectclass | Object }
```

Vous pouvez spécifier également Public (Visual Basic), Protected (Visual Basic), Friend (Visual Basic), **Protected Friend**, Private (Visual Basic), Shared (Visual Basic) ou Static (Visual Basic) dans la déclaration. Les exemples de déclarations suivants sont valides :

```
Private objA As Object
Static objB As System.Windows.Forms.Label
Dim objC As System.OperatingSystem
```

Liaison tardive et liaison anticipée

Parfois, la classe spécifique n'est pas connue jusqu'à l'exécution du code. Dans ce cas, vous devez déclarer la variable objet à l'aide du type de données **Object**. Cette opération crée une référence générale à tout type d'objet, et la classe spécifique est assignée au moment de l'exécution. Cette situation s'appelle une *liaison tardive*. La liaison tardive requiert un temps d'exécution supplémentaire. Elle limite également votre code aux méthodes et aux propriétés de la classe qui lui ont été assignées le plus récemment. Cette situation peut provoquer des erreurs d'exécution si votre code tente d'accéder aux membres d'une classe différente.

Lorsque vous connaissez la classe spécifique au moment de la compilation, vous devez déclarer la variable objet de telle sorte qu'elle appartienne à cette classe. Cette situation s'appelle une *liaison anticipée*. La liaison anticipée améliore les performances et garantit l'accès de votre code à toutes les méthodes et propriétés de la classe spécifique. Dans les exemples de déclarations précédents, si la variable `objA` utilise uniquement des objets de classe System.Windows.Forms.Label, vous devez spécifier `As System.Windows.Forms.Label` dans sa déclaration.

Avantages de la liaison anticipée

La déclaration d'une variable objet en tant que classe spécifique présente plusieurs avantages :

- Vérification automatique du type
- Accès garanti à tous les membres de la classe spécifique
- Prise en charge de Microsoft IntelliSense dans l'éditeur de code
- Lisibilité améliorée de votre code
- Réduction des erreurs dans votre code
- Les erreurs sont interceptées au moment de la compilation plutôt qu'au moment de l'exécution
- Exécution plus rapide du code

Accès aux membres des variables objets

Lorsque **Option Strict** a la valeur **On**, une variable objet ne peut accéder qu'aux méthodes et aux propriétés de la classe utilisée pour sa déclaration. L'exemple suivant illustre ce comportement :

```
' Option statements must precede all other source file lines.
Option Strict On
```

```
' Imports statement must precede all declarations in the source file.
Imports System.Windows.Forms
Public Sub accessMembers()
    Dim p As Object
    Dim q As System.Windows.Forms.Label
    p = New System.Windows.Forms.Label
    q = New System.Windows.Forms.Label
    Dim j, k As Integer
    ' The following statement generates a compiler ERROR.
    j = p.Left
    ' The following statement retrieves the left edge of the label in pixels.
    k = q.Left
End Sub
```

Dans cet exemple, `p` peut utiliser uniquement les membres de la classe **Object** elle-même, lesquels n'incluent pas la propriété `Left`. D'un autre côté, `q` a été déclaré comme étant de type **Label**, aussi il peut utiliser toutes les méthodes et les propriétés de la classe **Label** dans l'espace de noms System.Windows.Forms.

Flexibilité des variables objets

Lorsque vous travaillez avec des objets dans une hiérarchie d'héritage, vous pouvez choisir la classe à utiliser pour la déclaration de vos variables objets. Lors de ce choix, vous devez comparer la flexibilité de l'assignation d'objets à l'accès aux membres d'une classe. Par exemple, examinez la hiérarchie d'héritage menant à la classe System.Windows.Forms.Form :

Object

Component

Control

ScrollableControl

ContainerControl

Form

Supposons que votre application définisse une classe form appelée `specialForm` qui hérite de la classe **Form**. Vous pouvez déclarer une variable objet qui fait référence spécifiquement à `specialForm` comme le montre l'exemple suivant.

```
Public Class specialForm
    Inherits System.Windows.Forms.Form
    ' Insert code defining methods and properties of specialForm.
End Class
Dim nextForm As New specialForm
```

La déclaration de l'exemple précédent permet de limiter la variable `nextForm` aux objets de la classe `specialForm`, mais également de rendre toutes les méthodes et propriétés de `specialForm` disponibles pour `nextForm` ainsi que tous les membres de toutes les classes dont hérite `specialForm`.

Vous pouvez rendre une variable objet plus générale en la déclarant pour qu'elle soit du type **Form** comme le montre l'exemple suivant :

```
Dim anyForm As System.Windows.Forms.Form
```

La déclaration dans l'exemple précédent vous permet d'assigner tout formulaire dans votre application à `anyForm`. Toutefois, bien que `anyForm` puisse accéder à tous les membres de classe **Form**, il ne peut pas utiliser des méthodes supplémentaires ou des propriétés définies pour les formulaires spécifiques, tels que `specialForm`.

Tous les membres d'une classe de base sont disponibles pour les classes dérivées, mais les membres supplémentaires d'une classe dérivée ne sont pas disponibles pour la classe de base.

Voir aussi

Tâches

Comment : déclarer une variable objet et lui affecter un objet dans Visual Basic

Comment : accéder aux membres d'un objet

Référence

New (Visual Basic)

Option Strict, instruction

Concepts

Variables objet dans Visual Basic

Assignation des variables objets

Valeurs des variables objets

Programmation sans type dans Visual Basic

Comment : accéder aux membres d'un objet

Lorsque vous avez une variable objet qui fait référence à un objet, vous souhaitez souvent travailler avec les membres de cet objet, tels que ses méthodes, propriétés, champs et événements. Par exemple, une fois que vous avez créé un nouvel objet Form, vous souhaiterez peut-être définir sa propriété Text ou appeler sa méthode Focus.

Accès aux membres

Vous pouvez accéder aux membres d'un objet par le biais de la variable qui y fait référence.

Pour accéder aux membres d'un objet

- Utilisez l'opérateur d'accès aux membres (**.**) entre le nom de variable objet et le nom de membre.

```
currentText = newForm.Text
```

  Si le membre est Shared (Visual Basic), vous n'avez pas besoin d'une variable pour y accéder. Pour plus d'informations, consultez Comment : accéder aux membres partagés et non partagés d'un objet.

Accès aux membres d'un objet de type connu

Si vous connaissez le type d'un objet au moment de la compilation, vous pouvez utiliser la *liaison anticipée* pour une variable qui y fait référence.

Pour accéder aux membres d'un objet dont vous connaissez le type au moment de la compilation

1. Déclarez la variable objet comme étant du type de l'objet que vous envisagez d'assigner à la variable.

```
Dim extraForm As System.Windows.Forms.Form
```

   Avec **Option Strict On**, vous pouvez affecter uniquement des objets **Form** (ou des objets d'un type dérivé de **Form**) `extraForm`. Si vous avez défini une classe ou une structure avec une conversion **CType** étendue à **Form**, vous pouvez assigner également cette classe ou structure à `extraForm`.

2. Utilisez l'opérateur d'accès aux membres (**.**) entre le nom de variable objet et le nom de membre.

```
extraForm.Show()
```

   Vous pouvez accéder à toutes les méthodes et propriétés spécifiques à la classe **Form**, quel que soit le paramètre **Option Strict**.

Accès aux membres d'un objet de type inconnu

Si vous ne connaissez pas le type d'un objet au moment de la compilation, vous devez utiliser la *liaison tardive* pour toute variable qui y fait référence.

Pour accéder aux membres d'un objet dont vous ne connaissez pas le type au moment de la compilation

1. Déclarez la variable objet comme étant de type de données Object. (La déclaration d'une variable comme **Object** revient à la déclarer comme System.Object.)

```
Dim someControl As Object
```

   Avec **Option Strict On**, vous pouvez accéder uniquement aux membres qui sont définis sur la classe **Object**.

2. Utilisez l'opérateur d'accès aux membres (**.**) entre le nom de variable objet et le nom de membre.

```
someControl.GetType()
```

Pour être en mesure d'accéder aux membres de tout objet que vous assignez à la variable objet, vous devez définir **Option Strict Off**. Lorsque vous faites cela, le compilateur ne peut pas garantir qu'un membre donné est exposé par l'objet que vous assignez à la variable. Si l'objet n'expose pas un membre auquel vous essayez d'accéder, une exception MemberAccessException est générée.

Voir aussi

Référence

Object, type de données
Option Strict, instruction
Object
Form
MemberAccessException

Concepts

Variables objet dans Visual Basic
Déclaration des variables objets

Assignation des variables objets

Vous devez utiliser une instruction d'assignation normale pour assigner un objet à une variable objet. Vous pouvez assigner une expression d'objet ou le mot clé Nothing, comme le montre l'exemple suivant.

```
Dim thisObject As Object
' The following statement assigns an object reference.
thisObject = Form1
' The following statement discontinues association with any object.
thisObject = Nothing
```

Nothing signifie qu'il n'y a pas d'objet actuellement assigné à la variable.

Initialisation

Lorsque votre code commence à s'exécuter, vos variables objet sont initialisées avec la valeur **Nothing**. Les variables dont les déclarations incluent l'initialisation sont réinitialisées avec les valeurs que vous spécifiez quand les instructions de déclaration sont exécutées.

Vous pouvez inclure l'initialisation dans votre déclaration à l'aide du mot clé New. Les instructions de déclaration suivantes déclarent des variables objets `testUri` et `ver` et leur assignent des objets spécifiques. Chacune utilise l'un des constructeurs surchargés de la classe appropriée pour initialiser l'objet.

```
Dim testUri As New System.Uri("http://www.microsoft.com")
Dim ver As New System.Version(6, 1, 0)
```

Dissociation

La définition d'une variable objet à la valeur **Nothing** supprime l'association entre la variable et un objet spécifique. Cela vous empêche de modifier accidentellement l'objet en modifiant la variable. Cela vous permet également de déterminer si la variable objet pointe vers un objet valide, comme le montre l'exemple suivant.

```
If otherObject IsNot Nothing Then
    ' otherObject refers to a valid object, so your code can use it.
End If
```

Si l'objet référencé par votre variable se trouve dans une autre application, ce test ne permet pas de déterminer si cette application vient de se terminer ou d'invalider l'objet.

Une variable objet avec une valeur de **Nothing** est également appelée une *référence Null*.

Instance actuelle

L'*instance en cours* d'un objet est celle dans laquelle le code est en train de s'exécuter. Étant donné que la totalité du code s'exécute dans une procédure, l'instance en cours est celle dans laquelle la procédure a été appelée.

Le mot clé Me sert de variable objet faisant référence à l'instance en cours. Si une procédure n'est pas Shared (Visual Basic), elle peut utiliser le mot clé **Me** pour obtenir un pointeur vers l'instance en cours. Les procédures partagées ne peuvent pas être associées à une instance spécifique d'une classe.

L'utilisation de **Me** est particulièrement utile pour passer l'instance en cours vers une procédure d'un autre module. Par exemple, supposons que vous disposez de plusieurs documents XML et souhaitez leur ajouter un texte standard. L'exemple suivant définit une procédure le permettant.

```
Sub addStandardText(XmlDoc As System.Xml.XmlDocument)
    XmlDoc.CreateTextNode("This text goes into every XML document.")
End Sub
```

Chaque objet document XML peut ensuite appeler la procédure et passer son instance en cours sous forme d'argument. C'est ce qu'illustre l'exemple suivant.

```
addStandardText(Me)
```

Voir aussi

Tâches

Comment : déclarer une variable objet et lui affecter un objet dans Visual Basic

Comment : faire en sorte qu'une variable objet ne fasse pas référence à une instance

Concepts

Variables objet dans Visual Basic

Déclaration des variables objets

Valeurs des variables objets

Comment : déclarer une variable objet et lui affecter un objet dans Visual Basic

Pour déclarer une variable de type de données Object, spécifiez **As Object** dans une instruction Dim (Visual Basic). Assignez un objet à cette variable en insérant celui-ci après le signe égal (**=**) dans une instruction d'assignation ou une clause d'initialisation.

Exemple

L'exemple suivant déclare une variable **Object** et lui assigne l'instance de classe en cours.

```
      Dim thisObject As Object
thisObject = Me
```

Vous pouvez associer la déclaration et l'assignation en initialisant la variable dans le cadre de sa déclaration. L'exemple suivant équivaut au précédent.

```
Dim thisObject As Object = Me
```

Compilation du code

Cet exemple nécessite :

- une référence à l'espace de noms System ;
- une classe, une structure ou un module où placer l'instruction **Dim** ;
- une procédure dans laquelle placer l'instruction d'assignation.

Voir aussi

Référence

Object, type de données
Me
Dim, instruction (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic
Variables objet dans Visual Basic
Déclaration des variables objets

Comment : faire en sorte qu'une variable objet ne fasse pas référence à une instance

Vous pouvez dissocier une variable objet de toute instance d'objet en lui affectant la valeur Nothing (Visual Basic).

Pour dissocier une variable objet de toute instance d'objet

- Affectez à la variable la valeur **Nothing** dans une instruction d'assignation.

```
' Assume account is a defined class
Dim currentAccount As account
currentAccount = Nothing
```

Programmation fiable

Si votre code essaie d'accéder à un membre d'une variable objet qui a la valeur **Nothing**, une exception NullReferenceException se produit. Si vous affectez fréquemment à une variable objet la valeur **Nothing** ou s'il est possible que la variable ne soit pas initialisée, il peut être souhaitable de placer les accès aux membres dans un bloc **Try...Catch...Finally**.

Sécurité

Si vous utilisez une variable objet pour des objets qui contiennent des données confidentielles ou sensibles, vous pouvez affecter à la variable la valeur **Nothing** lorsque vous n'utilisez pas activement l'un de ces objets. Cela réduit le risque que du code malveillant accède aux données.

Voir aussi

Référence

Nothing (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)
Dépannage des exceptions : System.NullReferenceException
NullReferenceException

Concepts

Variables objet dans Visual Basic
Assignation des variables objets

Valeurs des variables objets

Une variable de type de données Object peut faire référence à des données de n'importe quel type. La valeur que vous stockez dans une variable **Object** est conservée quelque part en mémoire, alors que la variable contient un pointeur vers les données.

Fonctions classifieur d'objets

Visual Basic fournit des fonctions qui retournent des informations sur la référence d'une variable **Object**, comme l'illustre le tableau suivant.

Fonction	Retourne True si la variable objet fait référence à
IsArray, fonction (Visual Basic)	Un tableau de valeurs au lieu d'une valeur unique
IsDate, fonction (Visual Basic)	Une valeur Type de données date (Visual Basic) ou une chaîne pouvant être interprétée com me une valeur associée à la date et l'heure
IsDBNull, fonction	Un objet de type DBNull qui représente des données manquantes ou inexistantes
IsError, fonction	Un objet d'exception qui dérive de Exception
IsNothing, fonction	Nothing (Visual Basic) signifie qu'il n'y a pas d'objet actuellement assigné à la variable
IsNumeric, fonction (Visual Basic)	Un nombre ou une chaîne pouvant être interprété comme un nombre
IsReference, fonction	Un type référence (tel qu'une chaîne, un tableau, un délégué ou un type classe)

Vous pouvez utiliser ces fonctions pour éviter de soumettre une valeur non valide à une opération ou une procédure.

TypeOf, opérateur

Vous pouvez également utiliser TypeOf, opérateur (Visual Basic) pour déterminer si une variable objet fait actuellement référence à un type de données spécifique. L'expression **TypeOf**...**Is** prend la valeur **True** si le type d'exécution de l'opérande est dérivé du type spécifié ou qu'il implémente celui-ci.

L'exemple suivant utilise **TypeOf** sur les variables objets qui font référence aux types valeur et référence.

```
' The following statement puts a value type (Integer) in an Object variable.
Dim num As Object = 10
' The following statement puts a reference type (Form) in an Object variable.
Dim frm As Object = New Form()
If TypeOf num Is Long Then Debug.WriteLine("num is Long")
If TypeOf num Is Integer Then Debug.WriteLine("num is Integer")
If TypeOf num Is Short Then Debug.WriteLine("num is Short")
If TypeOf num Is Object Then Debug.WriteLine("num is Object")
If TypeOf frm Is Form Then Debug.WriteLine("frm is Form")
If TypeOf frm Is Label Then Debug.WriteLine("frm is Label")
If TypeOf frm Is Object Then Debug.WriteLine("frm is Object")
```

L'exemple précédent écrit les lignes suivantes dans la fenêtre de **débogage** :

```
num is Integer
num is Object
frm is Form
frm is Object
```

La variable objet `num` fait référence aux données de type **Integer**, et `frm` fait référence à un objet de la classe Form.

Tableaux d'objets

Vous pouvez déclarer et utiliser un tableau de variables **Object**. Cela peut vous être utile lorsque vous devez gérer plusieurs

types de données et classes d'objets. Tous les éléments d'un tableau doivent posséder le même type de données déclaré. En déclarant le type de données **Object**, vous pouvez stocker des objets et des instances de classes parmi d'autres types de données dans le tableau.

Voir aussi

Tâches

Comment : référencer l'instance en cours d'un objet
Comment : déterminer le type désigné par une variable objet
Comment : déterminer si deux objets sont liés
Comment : déterminer si deux objets sont identiques

Référence

Is (Visual Basic)

Concepts

Variables objet dans Visual Basic
Déclaration des variables objets
Assignation des variables objets
Types de données en Visual Basic

Comment : référencer l'instance en cours d'un objet

L'*instance en cours* d'un objet est celle dans laquelle le code est en train de s'exécuter.

Vous utilisez le mot clé Me pour faire référence à l'instance en cours.

Pour faire référence à l'instance en cours

- Utilisez le mot clé **Me** où vous utiliseriez normalement le nom d'une variable objet.

```
Me.ForeColor = System.Drawing.Color.Crimson
Me.Close()
```

  Bien que **Me** se comporte comme une variable objet, vous ne pouvez pas le déclarer ni lui assigner quoi que ce soit. **Me** fait toujours référence à l'instance en cours.

Voir aussi

Référence

Me

Concepts

Variables objet dans Visual Basic

Assignation des variables objets

Comment : déterminer le type désigné par une variable objet

Une variable objet contient un pointeur vers des données stockées ailleurs. Le type de ces données peut changer durant l'exécution. À tout moment, vous pouvez utiliser la méthode GetTypeCode pour déterminer le type d'exécution actuel ou le TypeOf, opérateur (Visual Basic) pour déterminer si le type d'exécution en cours est compatible avec un type spécifié.

Pour déterminer le type exact auquel une variable objet fait actuellement référence

1. Sur la variable objet, appelez la méthode GetType pour récupérer un objet System.Type.

   ```
   Dim myObject As Object
   myObject.GetType()
   ```

2. Sur la classe **System.Type**, appelez la méthode partagée **GetTypeCode** pour récupérer la valeur d'énumération TypeCode pour le type de l'objet.

   ```
   Dim myObject As Object
   Dim datTyp As Integer = Type.GetTypeCode(myObject.GetType())
   MsgBox("myObject currently has type code " & CStr(datTyp))
   ```

   Vous pouvez tester la valeur d'énumération **TypeCode** contre tout membre d'énumération digne d'intérêt, tel que Double.

Pour déterminer si le type d'une variable objet est compatible avec un type spécifié

- Utilisez l'opérateur **TypeOf** avec l'Is, opérateur (Visual Basic) pour tester l'objet avec une expression **TypeOf**...**Is**.

  ```
  If TypeOf objA Is System.Windows.Forms.Control Then
      MsgBox("objA is compatible with the Control class")
  End If
  ```

  L'expression **TypeOf**...**Is** retourne **True** si le type de l'objet à l'exécution est compatible avec le type spécifié.

  Le critère de compatibilité varie selon que le type spécifié est une classe, une structure ou une interface. En général, les types sont compatibles si l'objet est du même type, hérite du type spécifié ou l'implémente. Pour plus d'informations, consultez TypeOf, opérateur (Visual Basic).

Compilation du code

Notez que le type spécifié ne peut pas être une variable ni une expression. Il doit s'agir du nom d'un type défini, tel qu'une classe, une structure ou une interface. Cela inclut des types intrinsèques, tels que **Integer** et **String**.

Voir aussi

Référence

Object, type de données
GetType
System.Type
GetTypeCode
TypeCode

Concepts

Variables objet dans Visual Basic
Valeurs des variables objets

Comment : déterminer si deux objets sont liés

Vous pouvez comparer deux objets pour déterminer la relation qui existe éventuellement entre les classes à partir desquelles ils sont créés. La méthode IsInstanceOfType de la classe System.Type retourne **True** si la classe indiquée hérite de la classe active ou si le type actif représente une interface prise en charge par la classe indiquée.

Pour déterminer si un objet hérite de la classe ou de l'interface d'un autre objet

1. Appelez la méthode GetType sur l'objet que vous pensez être du type de base.
2. Sur l'objet **System.Type** retourné par **GetType**, appelez la méthode **IsInstanceOfType**.
3. Dans la liste d'arguments pour **IsInstanceOfType**, spécifiez l'objet que vous pensez être du type dérivé.

   IsInstanceOfType retourne **True** si son type d'argument hérite du type d'objet **System.Type**.

Exemple

L'exemple ci-dessous détermine si un objet représente une classe dérivée de la classe d'un autre objet.

```
Public Class baseClass
End Class
Public Class derivedClass : Inherits baseClass
End Class
Public Class testTheseClasses
    Public Sub seeIfRelated()
        Dim baseObj As Object = New baseClass()
        Dim derivedObj As Object = New derivedClass()
        Dim related As Boolean
        related = baseObj.GetType().IsInstanceOfType(derivedObj)
        MsgBox(CStr(related))
    End Sub
End Class
```

Notez l'emplacement inattendu des deux variables objets dans l'appel à **IsInstanceOfType**. Le type de base supposé est utilisé pour générer la classe **System.Type** et le type dérivé supposé est transmis sous forme d'argument à la méthode **IsInstanceOfType**.

Voir aussi

Tâches

Comment : déterminer si deux objets sont identiques

Référence

Object, type de données
GetType
System.Type
IsInstanceOfType

Concepts

Variables objet dans Visual Basic
Valeurs des variables objets

Comment : déterminer si deux objets sont identiques

Visual Basic considère deux objets comme étant identiques si leurs pointeurs sont les mêmes, à savoir, si les deux variables pointent vers la même instance de classe en mémoire. Vous souhaiterez peut-être effectuer cette comparaison, par exemple pour déterminer si l'instance actuelle (Me) est la même qu'une instance particulière, telle que `Form2`.

Visual Basic fournit deux opérateurs pour comparer des pointeurs. L'opérateur Is (Visual Basic) retourne **True** si les objets sont identiques et l'opérateur IsNot retourne **True** s'ils ne le sont pas.

Comment déterminer si deux objets sont identiques

Pour déterminer si deux objets sont identiques

1. Configurez une expression **Boolean** pour tester les deux objets.
2. Dans votre expression de test, utilisez l'opérateur **Is** avec les deux objets comme opérandes.

   Is retourne **True** si les objets pointent vers la même instance de classe.

Comment déterminer si deux objets ne sont pas identiques

Vous souhaitez parfois exécuter une action si les deux objets ne sont pas identiques, mais il peut être délicat de combiner **Not** et **Is**, par exemple `If Not obj1 Is obj2`. Dans ce cas, vous pouvez utiliser l'opérateur **IsNot**.

Pour déterminer si deux objets ne sont pas identiques

1. Configurez une expression **Boolean** pour tester les deux objets.
2. Dans votre expression de test, utilisez l'opérateur **IsNot** avec les deux objets comme opérandes.

   IsNot retourne **True** si les objets ne pointent pas vers la même instance de classe.

Exemple

L'exemple suivant teste des paires de variables **Object** pour vérifier si elles pointent vers la même instance de classe.

```
Dim objA, objB, objC as Object
objA = Me
objB = New System.Windows.Forms.Form()
objC = Me
MsgBox("objA different from objB? " & CStr(objA IsNot objB))
MsgBox("objA identical to objC? " & CStr(objA Is objC))
```

L'exemple précédent affiche la sortie suivante.

```
objA different from objB? True

objA identical to objC? True
```

Voir aussi

Tâches

Comment : déterminer si deux objets sont liés

Référence

Object, type de données
Is, opérateur (Visual Basic)
IsNot, opérateur

Concepts

Variables objet dans Visual Basic
Valeurs des variables objets

Dépannage des variables en Visual Basic

Cette page répertorie quelques problèmes courants qui peuvent se produire lors de l'utilisation des variables en Visual Basic.

Impossible d'accéder aux membres d'un objet

Si votre code essaie d'accéder à une propriété ou une méthode sur un objet, deux types d'erreur possibles peuvent se produire :

- Le compilateur peut générer un message d'erreur si vous déclarez la variable objet comme étant d'un type spécifique puis vous faites référence à un membre qui n'est pas défini par ce type.
- Une exception MemberAccessException runtime se produit lorsque l'objet assigné à une variable objet n'expose pas le membre que votre code essaie d'accéder. Dans le cas d'une variable de type de données Object, vous pouvez également obtenir cette exception si le membre n'a pas la valeur **Public**. Ceci s'explique par le fait que la liaison tardive autorise l'accès uniquement aux membres **Public**.

Lorsque l'instruction Option Strict affecte la valeur **On**, à la vérification de type, une variable objet ne peut accéder qu'aux méthodes et aux propriétés de la classe utilisée pour sa déclaration. L'exemple suivant illustre ce comportement :

VB

```
Option Strict On
```

VB

```
Dim p As Object = New System.Windows.Forms.Label
Dim q As System.Windows.Forms.Label = New System.Windows.Forms.Label
Dim j, k As Integer
' The following statement generates a compiler error.
j = p.Left
' The following statement retrieves the left edge of the label
' in pixels.
k = q.Left
```

Dans cet exemple, `p` peut utiliser uniquement les membres de la classe Object elle-même, lesquels n'incluent pas la propriété `Left`. D'un autre côté, `q` a été déclaré comme étant de type Label, aussi il peut utiliser toutes les méthodes et les propriétés de la classe **Label** dans l'espace de noms System.Windows.Forms.

Approche correcte

Pour pouvoir accéder à tous les membres d'un objet d'une classe particulière, déclarez la variable objet comme étant du type de cette classe, si possible. Sinon, si vous ne connaissez pas le type de l'objet au moment de la compilation, vous devez affecter **Option Strict** à **Off** et déclarer la variable comme étant du type de données Object. Cela permet d'assigner des objets de n'importe quel type à la variable, et vous devez faire en sorte que l'objet actuellement assigné soit d'un type acceptable. Pour ce faire, vous pouvez utiliser TypeOf, opérateur (Visual Basic).

D'autres composants ne peuvent pas accéder à votre variable

Les noms Visual Basic ne *respectent pas la casse*. Si deux noms ne se distinguent que par la casse, le compilateur les interprète comme un seul et même nom. Il considère, par exemple, que `ABC` et `abc` font référence au même élément déclaré.

Néanmoins, le Common Language Runtime (CLR) utilise des liaisons qui *respectent la casse*. En conséquence, lorsque vous générez un assembly ou une DLL et que vous les mettez à la disposition d'autres assemblys, vos noms respectent alors la casse. Par exemple, si vous définissez une classe avec un élément appelé `ABC` et que d'autres assemblys utilisent votre classe via le Common Language Runtime, ils doivent faire référence à l'élément sous la forme `ABC`. Si, par la suite, vous recompilez votre classe et que vous changez le nom de l'élément en `abc`, les autres assemblys qui utilisent votre classe ne peuvent plus accéder à cet élément. Dès lors, lorsque vous libérez une version mise à jour d'un assembly, évitez de modifier la casse des éléments publics.

Pour plus d'informations, consultez Common Language Runtime.

Approche correcte

Pour permettre à d'autres composants d'accéder à vos variables, considérez leurs noms comme s'ils respectaient la casse. Lorsque vous testez votre classe ou module, vérifiez que les autres assemblys se lient aux variables prévues. Une fois que vous avez publié un composant, ne modifiez pas les noms de variables existants, et notamment leurs casses.

Variable incorrecte utilisée

Lorsque plusieurs variables ont le même nom, le compilateur Visual Basic essaie de résoudre chaque référence à ce nom. Si les variables ont une portée différente, le compilateur résout une référence à la déclaration dont la portée est la plus petite. Si elles ont la même portée, la résolution échoue et le compilateur signale une erreur. Pour plus d'informations, consultez Résolution d'une référence lorsque plusieurs variables ont le même nom.

Approche correcte

Évitez d'utiliser des variables qui ont le même nom, mais une portée différente. Si vous utilisez d'autres assemblys ou projets, évitez d'utiliser autant que possible des noms définis dans ces composants externes. Si plusieurs variables ont le même nom, n'oubliez pas de qualifier chaque référence à celui-ci. Pour plus d'informations, consultez Comment : faire la distinction entre deux éléments portant le même nom.

Voir aussi

Tâches

Comment : accéder aux membres d'un objet
Comment : déterminer le type désigné par une variable objet

Concepts

Variables en Visual Basic
Déclaration de variable dans Visual Basic
Variables objet dans Visual Basic
Déclaration des variables objets
Valeurs des variables objets
Résolution d'une référence lorsque plusieurs variables ont le même nom
Noms d'éléments déclarés

Tableaux dans Visual Basic

Lorsque vous utilisez des tableaux, vous pouvez faire référence à plusieurs valeurs par un nom identique, en utilisant un nombre appelé un *index* ou un *indice* pour les distinguer les unes des autres. Les tableaux peuvent réduire et simplifier votre code, en vous permettant de créer des boucles qui traitent efficacement tout nombre d'éléments.

Dans cette section

Vue d'ensemble des tableaux dans Visual Basic

Définit les termes pertinents et expose des tableaux à titre d'exemple.

Dimensions du tableau dans Visual Basic

Explique le rang et les dimensions des tableaux.

Tableaux multidimensionnels dans Visual Basic

Examine les tableaux qui affichent plusieurs dimensions.

Tableaux en escalier dans Visual Basic

Décrit des tableaux de tableaux, également connus sous le nom de tableaux à plusieurs dimensions.

Types de données tableau dans Visual Basic

Décrit les facteurs qui déterminent le type de données d'un tableau.

Taille du tableau dans Visual Basic

Décrit les aspects à prendre en compte concernant la taille d'un tableau et chacune de ses dimensions.

Utilisation des tableaux et les éléments de tableau

Décrit le tri des tableaux, l'obtention ou le placement d'une valeur de tableau, etc.

Collections en tant qu'alternative aux tableaux

Compare le stockage des éléments dans une collection à leur stockage dans un tableau.

Dépannage des tableaux

Aborde certains problèmes courants qui surviennent lors de l'utilisation des tableaux.

Rubriques connexes

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Propose une vue d'ensemble des fonctionnalités nouvelles de Visual Basic 2005.

Programmation orientée objet dans Visual Basic

Traite de la programmation orientée objet.

Vue d'ensemble des tableaux dans Visual Basic

Un *tableau* est un ensemble de valeurs liées de manière logique, telles que le nombre d'étudiants de chaque niveau scolaire dans un lycée.

Un tableau vous permet de faire référence à ces valeurs connexes par le même nom et d'utiliser un numéro, appelé *index* ou *indice*, pour les distinguer. Les valeurs individuelles portent le nom d'*éléments* du tableau. Elles sont contiguës de l'index 0 jusqu'à la valeur d'index la plus élevée.

Exemple

L'exemple suivant déclare une variable tableau devant contenir le nombre d'étudiants de chaque niveau scolaire dans un lycée.

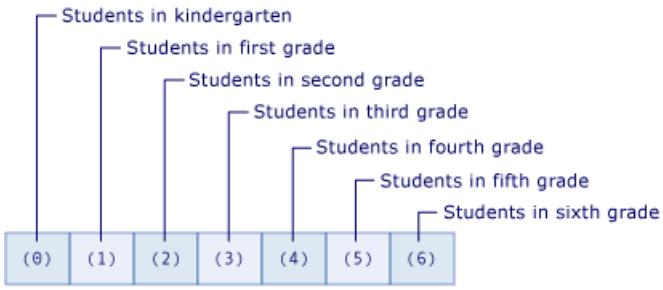
```
Dim students(6) As Integer
```

Le tableau `students` de l'exemple précédent contient 7 éléments. Les index des éléments varient de 0 à 6. Il est plus simple d'avoir ce tableau que de déclarer 7 variables différentes.

L'illustration suivante représente le tableau `students`. Pour chaque élément du tableau :

- l'index de l'élément représente le niveau scolaire (l'index 0 représente le jardin d'enfants) ;
- la valeur contenue dans l'élément représente le nombre d'étudiants dans ce niveau scolaire.

Éléments du tableau « students »



L'exemple suivant montre comment faire référence aux premier, deuxième et dernier éléments du tableau `students`.

```
Dim kindergarten As Integer = students(0)
Dim firstGrade As Integer = students(1)
Dim sixthGrade As Integer = students(6)
MsgBox("Students in kindergarten = " & CStr(kindergarten))
MsgBox("Students in first grade = " & CStr(firstGrade))
MsgBox("Students in sixth grade = " & CStr(sixthGrade))
```

Vous pouvez faire référence au tableau dans son ensemble en utilisant uniquement le nom de variable tableau sans index.

Types tableau et autres types

Types de données

Chaque tableau a un type de données, mais celui-ci n'est pas identique au type de données de ses éléments. Par exemple, le tableau `students` de l'exemple précédent est de type **Integer()**, tandis que chacun de ses éléments est de type **Integer**. La notation **Integer()** signifie que le tableau contient des éléments **Integer**. Pour plus d'informations, consultez Types de données tableau dans Visual Basic.

Chaque tableau hérite de la classe System.Array et vous pouvez déclarer une variable comme étant de type **Array**, mais vous ne pouvez pas créer de tableau de type **Array**. De plus, l'ReDim, instruction (Visual Basic) ne peut pas fonctionner sur une variable déclarée comme type **Array**. Pour ces raisons, et pour la sécurité de type, il est recommandé de déclarer chaque tableau en tant que type spécifique, tel que **Integer** dans l'exemple précédent.

Dimensions d'un tableau

Le tableau `students` de l'exemple précédent utilise un index et est *unidimensionnel*. Un tableau qui utilise plusieurs index ou indices est *multidimensionnel*.

Un autre genre de tableau est un tableau qui contient d'autres tableaux comme éléments. On appelle cela un *tableau de tableaux* ou un *tableau en escalier*. Un tableau en escalier peut être unidimensionnel ou multidimensionnel, comme peuvent l'être ses éléments.

Types opposés

Une variable qui contient une valeur unique est appelée variable *scalaire*.

Un tableau est différent d'une *collection*. Pour plus d'informations, consultez Collections en tant qu'alternative aux tableaux.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Dimensions du tableau dans Visual Basic
Tableaux multidimensionnels dans Visual Basic
Tableaux en escalier dans Visual Basic
Types de données tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : déclarer une variable tableau

Vous pouvez déclarer des variables tableau de la même façon que les autres variables, à l'aide de l'instruction **Dim**. Le nom de la variable est suivi d'une ou plusieurs paires de parenthèses pour indiquer qu'elle contient un tableau et non une variable *scalaire* (une variable contenant une valeur unique).

Déclaration de variables tableau

Pour déclarer une variable tableau unidimensionnelle

- Dans votre déclaration, ajoutez une paire de parenthèses après le nom de la variable. L'exemple suivant déclare une variable devant contenir un tableau unidimensionnel avec des éléments du Double, type de données (Visual Basic).

  ```
  Dim cargoWeights() As Double
  ```

  L'exemple précédent déclare une variable tableau, mais ne lui affecte pas de tableau. Vous devez tout de même créer un tableau unidimensionnel, l'initialiser et l'assigner à `cargoWeights`.

Pour déclarer une variable tableau multidimensionnelle

- Dans votre déclaration, ajoutez une paire de parenthèses après le nom de la variable et insérez des virgules dans les parenthèses pour séparer les dimensions. L'exemple suivant déclare une variable devant contenir un tableau à quatre dimensions avec des éléments du Short, type de données (Visual Basic).

  ```
  Dim atmospherePressures(,,,) As Short
  ```

  L'exemple précédent déclare une variable tableau, mais ne lui affecte pas de tableau. Vous devez tout de même créer un tableau à quatre dimensions, l'initialiser et l'assigner à `atmospherePressures`.

Pour déclarer une variable tableau en escalier

- Dans votre déclaration, ajoutez le même nombre de paires de parenthèses après le nom de la variable que de niveaux de tableaux imbriqués. L'exemple suivant déclare une variable devant contenir un tableau de tableaux de tableaux (un tableau dont chaque élément est un tableau, dont chaque élément est un tableau), le tableau le plus profond contenant des éléments du Byte, type de données (Visual Basic).

  ```
  Dim inquiriesByYearMonthDay()()() As Byte
  ```

  L'exemple précédent déclare une variable tableau, mais ne lui affecte pas de tableau. Vous devez tout de même créer un tableau de tableaux de tableaux, l'initialiser et l'assigner à `inquiriesByYearMonthDay`.

Voir aussi

Tâches

Comment : créer un tableau
Comment : créer un tableau à plusieurs dimensions
Comment : créer un tableau de tableaux
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic
Tableaux en escalier dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : créer un tableau

Un tableau est un objet, que vous pouvez donc créer à l'aide d'une clause New (Visual Basic) et assigner à une variable tableau. Vous pouvez effectuer cette opération dans le cadre de la déclaration de tableau ou dans une instruction d'assignation ultérieure.

Pour créer un tableau dans l'instruction de déclaration de tableau

- Dans votre déclaration, ajoutez une clause **New** après le nom de variable et ses parenthèses. L'exemple suivant déclare une variable destinée à contenir un tableau d'éléments de Type de données date (Visual Basic), crée le tableau et l'assigne à la variable.

```
Dim validDates() As Date = New Date() {}
```

Après l'exécution de cette instruction, la longueur du tableau dans la variable `validDates` est de 0.

Remarque

La clause **New** doit spécifier le nom de type, suivie de parenthèses, puis d'accolades, **{}**. Les parenthèses ne représentent pas un appel à un constructeur de tableau. Au contraire, elles indiquent que le type d'objet est un type tableau. Les accolades fournissent des valeurs d'initialisation. Le compilateur requiert les accolades même si vous ne fournissez pas de valeurs. Par conséquent, la clause **New** doit inclure à la fois les parenthèses et les accolades, même si elles sont vides.

Pour créer un tableau dans une instruction d'assignation séparée

- Utilisez une instruction d'assignation ultérieure avec une clause **New**. L'exemple suivant déclare une variable destinée à contenir un tableau d'éléments de Integer, type de données (Visual Basic), crée le tableau et l'assigne à la variable dans une autre déclaration.

```
Dim scores() As Integer
scores = New Integer() {}
```

Après l'exécution de ces instructions, la longueur du tableau dans la variable `scores` est de 0.

- ou -

- Utilisez ReDim, instruction (Visual Basic) pour créer un tableau et initialiser sa longueur.

```
ReDim scores(4)
```

Après l'exécution de cette instruction, la longueur du tableau dans la variable `scores` est de 5, et tous les éléments ont des valeurs par défaut.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau à plusieurs dimensions
Comment : créer un tableau de tableaux
Comment : créer un tableau avec des types d'élément mixtes
Comment : créer un tableau sans éléments
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : créer un tableau avec des types d'élément mixtes

Vous devez déclarer uniquement un type de données pour un tableau, et tous ses éléments doivent correspondre à ce type de données. Normalement cette limitation est souhaitable puisque tous les éléments sont étroitement liés entre eux et possèdent des valeurs de types semblables. Toutefois, les éléments ne sont pas toujours étroitement liés entre eux ou n'ont pas de valeurs semblables. Dans ce cas, vous pouvez déclarer les éléments de tableau pour qu'ils soient de type de données Object, et les éléments individuels peuvent pointer sur différents types de données, tels que des nombres, des caractères, des chaînes, des objets et d'autres tableaux.

Pour créer un tableau avec des éléments de types de données différents

- Déclarez le tableau comme **Object**. L'exemple suivant déclare une variable destinée à contenir un tableau d'éléments **Object**, crée le tableau et l'assigne à la variable.

  ```
  Dim mixedTypes As Object() = New Object() {}
  ```

  Lorsque vous utilisez le type de données **Object**, n'oubliez pas que les performances ne sont pas aussi efficaces que lorsque vous utilisez un type de données plus spécifique. En effet, l'exécution doit convertir les données entre son type de données naturel et **Object** à l'aide d'opérations appelées des conversion *boxing* et *unboxing*. Ce traitement supplémentaire peut nuire à vos performances s'il a lieu souvent.

Pour accéder aux éléments de types de données différents dans un tableau

- Lisez ou écrivez les éléments normalement. Vous pouvez stocker et extraire un élément de n'importe quel type de données dans un tableau **Object**.

  L'exemple suivant montre comment insérer des informations de types de données différents dans un tableau **Object**. Il stocke des informations d'employé dans le tableau dans la variable `employeeData`.

  ```
  Dim employeeData(3) As Object
  employeeData(0) = "Alex Hankin"
  employeeData(1) = "4242 Maple Street"
  employeeData(2) = 48
  employeeData(3) = #8/23/1956#
  ```

  Pour extraire des informations de types de données différents d'un tableau **Object**, vous pouvez convertir un élément en type de données approprié, comme le montre l'exemple suivant.

  ```
  Dim age As Integer = CInt(employeeData(2))
  Dim birthDate as Date = CDate(employeeData(3))
  ```

Dans une situation où les éléments ne sont pas semblables ou ne sont pas liés entre eux, une autre possibilité consiste à les mettre dans une collection au lieu d'un tableau **Object**. Pour plus d'informations, consultez Collections en tant qu'alternative aux tableaux.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : créer un tableau à plusieurs dimensions
Comment : créer un tableau de tableaux
Comment : créer un tableau sans éléments
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Collections en tant qu'alternative aux tableaux

Autres ressources

Comment : créer un tableau sans éléments

Un tableau sans éléments est également appelé un *tableau de longueur zéro*. Une variable contenant un tableau de longueur zéro n'a pas la valeur **Nothing**.

Vous devrez peut-être créer un tableau de longueur zéro dans les circonstances suivantes :

- Votre code doit accéder aux membres de la classe Array, tels que Length ou Rank, ou appeler une fonction Visual Basic telle que UBound, fonction (Visual Basic), sans risquer une exception NullReferenceException.
- Vous devez veiller à une plus grande simplification du code utilisateur sans avoir à vérifier **Nothing** comme un cas particulier.
- Votre code interagit avec une interface de programmation d'applications (API) qui requiert que vous passiez un tableau de longueur zéro à une ou plusieurs procédures, ou qui retourne un tableau de longueur zéro d'une ou plusieurs procédures.

Pour créer un tableau sans éléments

- Déclarez l'une des dimensions du tableau comme étant égale à -1. L'exemple suivant déclare une variable destinée à stocker un tableau composé d'éléments de String, type de données (Visual Basic), mais définit initialement celui-ci comme étant vide.

```
Dim twoDimensionalStrings(-1, 3) As String
```

Après l'exécution de cette instruction, le tableau dans la variable `twoDimensionalStrings` est à deux dimensions avec une longueur égale à 0. Il est vide, mais il existe néanmoins. Par conséquent, une variable qui pointe vers ce tableau n'a pas pour valeur **Nothing (rien)**. Par la suite, vous pouvez créer un tableau non vide et l'assigner à `twoDimensionalStrings`.

Par opposition, l'exemple suivant déclare une variable tableau qui ne pointe pas initialement vers un tableau.

```
Dim twoDimStrings( , ) As String
```

Contrairement à `twoDimensionalStrings` dans l'exemple précédent, la variable `twoDimStrings` a la valeur **Nothing**.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : créer un tableau à plusieurs dimensions
Comment : créer un tableau de tableaux
Comment : créer un tableau avec des types d'élément mixtes
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : initialiser une variable tableau

Vous pouvez initialiser un tableau en même temps que vous le créez, dans le cadre de la clause New (Visual Basic). Vous pouvez également l'initialiser dans des instructions d'assignation ultérieures.

Les aspects d'un tableau que vous pouvez initialiser sont les suivants :

- les limites supérieures d'index, qui spécifient les longueurs des dimensions du tableau ;
- les valeurs de tout ou partie des éléments du tableau.

Vous pouvez initialiser l'un sans initialiser l'autre. Toutefois, si vous fournissez des valeurs d'éléments mais pas des limites supérieures, le nombre de valeurs que vous fournissez détermine les limites supérieures.

Pour initialiser un tableau dans la clause New au moment de la création

- Dans la clause **New**, spécifiez la limite supérieure d'index à l'intérieur des parenthèses et fournissez les valeurs d'éléments à l'intérieur des crochets (**{}**). L'exemple suivant déclare, crée et initialise une variable devant contenir un tableau avec des éléments du Char, type de données (Visual Basic), en spécifiant la limite supérieure et les valeurs.

  ```
  Dim testChars As Char() = New Char(2) {"%"c, "&"c, "@"c}
  ```

  Suivant l'exécution de cette instruction, le tableau dans la variable `testChars` a la longueur 3, avec les éléments de l'index 0 à l'index 2 contenant des valeurs initialisées. Si vous fournissez à la fois la limite supérieure et les valeurs, vous devez inclure une valeur pour chaque élément de l'index 0 jusqu'à la limite supérieure.

  Notez le caractère de type de littéral **c** , qui force un littéral de caractère au type de données **Char**. Sans caractère de type, un littéral compris entre guillemets doubles (**" "**) assume par défaut le type **String**.

  Vous n'êtes pas obligé de spécifier la limite supérieure d'index si vous fournissez des valeurs d'éléments dans la clause **New**. L'exemple suivant déclare, crée et initialise une variable devant contenir un tableau avec des éléments du Boolean, type de données (Visual Basic), en spécifiant uniquement les valeurs d'éléments.

  ```
  Dim answers As Boolean() = New Boolean() {True, True, False, True}
  ```

  Suivant l'exécution de cette instruction, le tableau dans la variable `answers` a la longueur 4, avec les éléments des index 0 à 3 contenant des valeurs initialisées.

Vous pouvez initialiser la limite supérieure d'index sans initialiser d'éléments. Si vous créez un tableau de cette manière, vous devez utiliser une instruction d'assignation ultérieure pour initialiser chaque valeur d'élément.

Pour initialiser un tableau dans des instructions d'assignation ultérieures

1. Spécifiez la limite supérieure d'index dans la déclaration de variable tableau.
2. Utilisez une ou plusieurs instructions d'assignation, chacune assignant une valeur à l'un des éléments du tableau. L'exemple suivant déclare et crée une variable devant contenir un tableau avec des éléments du String, type de données (Visual Basic) et fournit deux valeurs d'éléments dans des instructions ultérieures.

   ```
   Dim comments(30) As String
   comments(0) = "This is the first comment."
   comments(5) = "This is the sixth comment."
   ```

   Suivant l'exécution de ces instructions, le tableau dans la variable `comments` a la longueur 31, avec les éléments des index 0 et 5 contenant des valeurs initialisées et les 29 autres éléments contenant des valeurs par défaut. Si vous initialisez un tableau de cette manière, vous pouvez initialiser certains éléments et en ignorer d'autres.

- ou -

- Utilisez l'ReDim, instruction (Visual Basic) pour initialiser la longueur du tableau.

```
ReDim comments(5)
```

Suivant l'exécution de cette instruction, le tableau dans la variable `comments` a la longueur 6 et tous les éléments contiennent des valeurs par défaut.

Remarque

Vous pouvez initialiser la limite supérieure d'index à un seul emplacement. Si vous spécifiez une limite supérieure dans les parenthèses qui suivent le nom de variable tableau, vous ne pouvez pas utiliser de clause **New**. Si vous spécifiez une limite supérieure dans les parenthèses dans la clause **New**, vous devez laisser vides les parenthèses qui suivent le nom de variable.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser un tableau multidimensionnel
Comment : initialiser un tableau en escalier
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Dimensions du tableau dans Visual Basic

Une *dimension* est une direction dans laquelle vous pouvez varier les caractéristiques des éléments d'un tableau. Un tableau qui contient le total des ventes pour chaque jour du mois possède une dimension (le jour du mois). Un tableau qui contient le total des ventes par service pour chaque jour du mois possède deux dimensions (le numéro du service et le jour du mois). Le nombre de dimensions que possède un tableau s'appelle un *rang*.

Utilisation des dimensions

Spécifiez un élément d'un tableau en fournissant un *index* ou un *indice* pour chacune de ses dimensions. Les éléments sont contigus pour chaque dimension, de l'index 0 à l'index le plus élevé de cette dimension.

Les illustrations suivantes montrent la structure conceptuelle de tableaux possédant des rangs différents. Chaque élément dans les illustrations affiche les valeurs d'index qui y accèdent. Par exemple, vous pouvez accéder au premier élément de la deuxième ligne du tableau à deux dimensions en spécifiant les index `(1, 0)`.

Tableau unidimensionnel

Tableau à deux dimensions

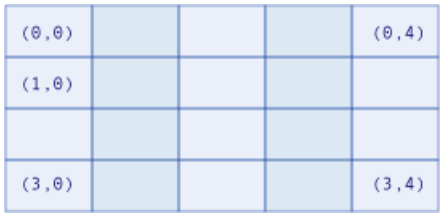
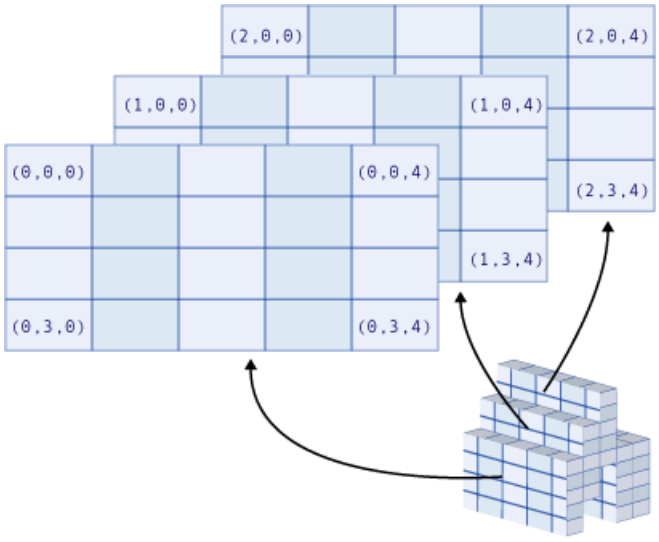


Tableau à trois dimensions



Unidimensionnel

De nombreux tableaux ont une seule dimension, telle que le nombre de personnes pour chaque âge. La seule condition pour spécifier un élément est l'âge pour lequel cet élément possède le compte. Par conséquent, ce type de tableau utilise un seul index. L'exemple suivant déclare une variable destinée à contenir un *tableau unidimensionnel* de comptes de personnes âgées de 0 à 120 ans.

```
Dim ageCounts(120) As UInteger
```

Deux dimensions

Certains tableaux ont deux dimensions, telles que le nombre de bureaux à chaque étage dans chaque bâtiment d'un campus. La spécification d'un élément requiert à la fois le numéro du bâtiment et de l'étage, et chaque élément possède le compte pour

cette combinaison de bâtiment et d'étage. Par conséquent, ce type de tableau utilise deux index. L'exemple suivant déclare une variable destinée à contenir un *tableau à deux dimensions* de comptes de bureau, pour des bâtiments allant de 0 à 40 et les étages de 0 à 5.

```
Dim officeCounts(40, 5) As Byte
```

Un tableau à deux dimensions est également appelé *tableau rectangulaire*.

Trois dimensions

Certains tableaux ont trois dimensions, telles que des valeurs dans un espace tridimensionnel. Ce type de tableau utilise trois index qui, dans le cas présent, correspondent aux coordonnées *x*, *y* et *z* de l'espace physique. L'exemple suivant déclare une variable destinée à contenir un *tableau à trois dimensions* de températures de l'air à plusieurs endroits dans un volume tridimensionnel.

```
Dim airTemperatures(99, 99, 24) As Single
```

Plus de trois dimensions

Bien qu'un tableau puisse avoir jusqu'à 32 dimensions, il est rare d'en avoir plus de trois.

Remarque

Lorsque vous ajoutez des dimensions à un tableau, le stockage total nécessaire pour le tableau augmente considérablement. Utilisez donc les tableaux multidimensionnels avec prudence.

Utilisation de dimensions différentes

Supposons que vous souhaitez effectuer le suivi des montants des ventes pour chaque jour du mois en cours. Vous pouvez déclarer un tableau unidimensionnel avec 31 éléments, un pour chaque jour du mois, comme le montre l'exemple suivant.

```
Dim salesAmounts(30) As Double
```

Supposons maintenant que vous souhaitez effectuer le suivi des mêmes informations non seulement pour chaque jour du mois, mais également pour chaque mois de l'année. Vous pouvez déclarer un tableau à deux dimensions avec 12 lignes (pour les mois) et 31 colonnes (pour les jours), comme le montre l'exemple suivant.

```
Dim salesAmounts(11, 30) As Double
```

Maintenant supposons que vous décidez que votre tableau stockera des informations pendant plus d'une année. Si vous souhaitez effectuer le suivi des montants des ventes sur 5 ans, vous pouvez déclarer un tableau à trois dimensions composé de 5 couches, de 12 lignes et de 31 colonnes, comme le montre l'exemple suivant.

```
Dim salesAmounts(4, 11, 30) As Double
```

Étant donné que chaque index varie de 0 à son maximum, notez que chaque dimension de `salesAmounts` est déclarée avec une valeur inférieure de 1 à la longueur requise pour cette dimension. Notez également que la taille du tableau augmente avec chaque nouvelle dimension. Les trois tailles dans les exemples précédents sont respectivement 31, 372 et 1 860 éléments.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic
Tableaux en escalier dans Visual Basic
Types de données tableau dans Visual Basic

Autres ressources

Comment : spécifier une limite inférieure égale à zéro dans un tableau

Si vous déclarez un tableau, vous pouvez spécifier la limite inférieure de chaque dimension à l'aide du caractère zéro (0) et du mot clé **To**. Cela ne modifie pas la limite inférieure requise, mais la lecture de votre code peut en être facilitée.

Pour spécifier explicitement la limite inférieure égale à zéro dans un tableau

1. Déclarez normalement le tableau.
2. À l'intérieur des parenthèses, ajoutez `0 To` devant la limite supérieure pour chaque dimension.

```
Public Sub declarelowerbounds()
    Dim monthtotal(0 To 11) As Double
    Dim cell(0 To 39, 0 To 19) As Integer
    MsgBox("Total number of elements:" _
        & vbCrLf & "monthtotal (0 To 11) length " & CStr(monthtotal.Length) _
        & vbCrLf & "cell (0 To 39, 0 To 19) length " & CStr(cell.Length))
End Sub
```

La limite inférieure doit toujours être 0, mais votre code peut être plus lisible lorsque vous la déclarez explicitement. La spécification des deux limites rappelle également au lecteur que la limite inférieure est 0.

Autre forme de création de tableau. Vous pouvez créer un tableau sans utiliser la clause Dim, instruction (Visual Basic) ou New (Visual Basic) Par exemple, vous pouvez appeler la méthode CreateInstance, ou un autre composant peut passer à votre code un tableau créé de cette manière. Ce type de tableau peut avoir des limites inférieures différentes de 0. Vous pouvez toujours tester la limite inférieure d'une dimension à l'aide de la méthode GetLowerBound ou à l'aide de LBound, fonction (Visual Basic).

Voir aussi

Tâches

Dépannage des tableaux

Concepts

Dimensions du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : déterminer le nombre de dimensions d'un tableau

La propriété **Rank** d'un tableau retourne son *rang* (le nombre de dimensions).

Pour déterminer le rang d'un tableau

- Lisez la propriété Rank du tableau. Ne faites pas suivre le nom de tableau par des parenthèses.

```
Dim thisThreeDimArray(,,) As Byte = New Byte(4, 4, 4) {}
MsgBox("Rank of thisThreeDimArray is " & CStr(thisThreeDimArray.Rank))
```

  L'appel **MsgBox** affiche le message "Rank of thisThreeDimArray is 3".

Le rang d'un tableau fait partie de son type de données. Pour cette raison, vous ne pouvez pas modifier le rang, même avec une instruction **ReDim**.

Voir aussi

Tâches

Comment : spécifier une limite inférieure égale à zéro dans un tableau
Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Dépannage des tableaux

Concepts

Dimensions du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Tableaux multidimensionnels dans Visual Basic

Un tableau peut avoir une ou plusieurs dimensions. S'il en a plusieurs, il porte le nom de tableau *multidimensionnel*. Notez qu'un tableau multidimensionnel n'est pas la même chose qu'un tableau *en escalier*, qui a d'autres tableaux comme éléments.

Dimensions et taille

La *dimensionnalité* ou le *rang* d'un tableau correspond au numéro d'index utilisé pour identifier un élément individuel. Vous pouvez spécifier jusqu'à 32 dimensions, bien qu'il soit rare d'en spécifier plus de trois. L'exemple suivant déclare une variable de tableau à deux dimensions et une variable de tableau à trois dimensions.

```
Dim populations(200, 3) As Long
Dim matrix(5, 15, 10) As Single
```

Le nombre total d'éléments est le produit des longueurs de toutes les dimensions. Dans l'exemple précédent, `populations` a un total de 804 éléments (201 x 4) et `matrix` a 1 056 éléments (6 x 16 x 11). Chaque index varie de 0 jusqu'à la longueur spécifiée pour sa dimension.

Un tableau à deux dimensions porte également le nom de *tableau rectangulaire*.

Remarque

Lorsque vous ajoutez des dimensions à un tableau, le stockage total nécessaire pour le tableau augmente considérablement. Utilisez donc les tableaux multidimensionnels avec prudence.

Membres de classe de tableau

Tous les tableaux héritent de la classe Array dans l'espace de noms System ; vous pouvez accéder aux méthodes et propriétés de **Array** de chaque tableau. Les membres suivants de **Array** peuvent être utiles :

- La propriété Rank retourne le rang du tableau (nombre de dimensions).
- La méthode GetLength retourne la longueur de la dimension spécifiée.
- La méthode GetUpperBound retourne la valeur d'index la plus élevée pour la dimension spécifiée. La valeur d'index la moins élevée pour chaque dimension est toujours 0.
- La propriété Length retourne le nombre total d'éléments dans le tableau.
- La méthode System.Array.Sort trie les éléments d'un tableau unidimensionnel.

Notez que **GetLength** et **GetUpperBound** prennent un argument basé sur 0 pour la dimension que vous spécifiez.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau à plusieurs dimensions
Comment : initialiser un tableau multidimensionnel
Comment : initialiser un tableau en escalier
Comment : initialiser un tableau multidimensionnel
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Dimensions du tableau dans Visual Basic
Tableaux en escalier dans Visual Basic
Taille du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : créer un tableau à plusieurs dimensions

Un tableau qui utilise plusieurs index est appelé un tableau *multidimensionnel*. Comme pour un tableau unidimensionnel, créez le tableau à l'aide d'une clause New (Visual Basic) et assignez-le à la variable tableau. Vous pouvez effectuer cette opération dans le cadre de la déclaration de tableau ou dans une instruction d'assignation suivante.

Pour créer un tableau multidimensionnel

1. Placez le nombre approprié de virgules à l'intérieur des parenthèses qui suivent le nom de la variable. Vous devez avoir une virgule de moins que le nombre de dimensions.

2. Placez le même nombre de virgules à l'intérieur des parenthèses dans la clause **New**. Vous n'avez pas besoin de virgules à l'intérieur des accolades (**{}**) si vous ne fournissez pas de valeurs d'élément.

   L'exemple suivant déclare une variable destinée à stocker un tableau à deux dimensions contenant des éléments du Double, type de données (Visual Basic), crée le tableau et l'assigne à la variable.

   ```
   Dim weights(,) As Double = New Double(,) {}
   ```

   Après l'exécution de cette instruction, la longueur du tableau dans la variable `weights` est de 0.

   Remarque

   Lorsque vous ajoutez des dimensions à un tableau, le stockage total nécessaire pour le tableau augmente considérablement. Utilisez donc les tableaux multidimensionnels avec prudence.

Pour travailler efficacement avec un tableau multidimensionnel

- Insérez-le dans une boucle **For** imbriquée.

  L'exemple suivant initialise chaque élément dans `matrix` avec une valeur comprise entre 0 et 99, en fonction de l'emplacement de l'élément dans le tableau.

  ```
  Dim matrix(9, 9) As Double
  Dim maxDim0 As Integer = UBound(matrix, 1)
  Dim maxDim1 As Integer = UBound(matrix, 2)
  For i As Integer = 0 To maxDim0
      For j As Integer = 0 To maxDim1
          matrix(i, j) = (i * 10) + j
      Next j
  Next i
  ```

  Un tableau multidimensionnel n'est pas le même qu'un tableau *en escalier*. Pour plus d'informations, consultez Comment : créer un tableau de tableaux.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : créer un tableau avec des types d'élément mixtes
Comment : créer un tableau sans éléments
Comment : initialiser un tableau multidimensionnel
Dépannage des tableaux

Référence

For...Next, instruction (Visual Basic)

Concepts

Tableaux multidimensionnels dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : initialiser un tableau multidimensionnel

Vous pouvez initialiser une variable de tableau multidimensionnel de la même manière qu'un tableau unidimensionnel, hormis le fait que vous devez tenir compte de chaque dimension.

Pour initialiser une variable de tableau multidimensionnel

- Dans la déclaration de variable tableau, spécifiez chaque limite supérieure d'index à l'intérieur des parenthèses, séparées par des virgules. L'exemple suivant déclare et crée une variable devant contenir un tableau à deux dimensions avec des éléments du type de données Short (Visual Basic), en spécifiant uniquement les limites supérieures.

```
Dim sizes(1, 1) As Short
```

Suivant l'exécution de cette instruction, le tableau dans la variable `sizes` a un total de quatre éléments, aux index (0, 0), (0, 1), (1, 0) et (1, 1), contenant des valeurs par défaut. Si vous créez un tableau de cette manière, vous devez utiliser une instruction d'assignation ultérieure pour assigner chaque valeur d'élément.

- ou -

- Suivez la déclaration avec un signe égal (**=**) et une clause New (Visual Basic). Dans la clause **New**, répétez le type de données d'élément, spécifiez les limites supérieures d'index entre parenthèses et fournissez des accolades vides (**{}**). L'exemple suivant déclare et crée une variable devant contenir un tableau à trois dimensions avec des éléments du type de données **Short**, en spécifiant uniquement les limites supérieures.

```
Dim replyCounts(,,) As Short = New Short(2, 1, 2) {}
```

Après l'exécution de cette instruction, le tableau dans la variable `replyCounts` possède18 éléments contenant des valeurs par défaut. Si vous créez un tableau de cette manière, vous devez utiliser une instruction d'assignation ultérieure pour assigner chaque valeur d'élément.

Remarque

Vous pouvez initialiser les limites supérieures d'index à un seul emplacement. Si vous spécifiez des limites supérieures dans les parenthèses qui suivent le nom de variable tableau, vous ne pouvez pas utiliser de clause **New**. Si vous spécifiez des limites supérieures dans les parenthèses dans la clause **New**, vous devez laisser vides les parenthèses qui suivent le nom de variable.

- ou -

- Dans la clause **New**, spécifiez chaque limite supérieure d'index à l'intérieur des parenthèses et fournissez les valeurs d'éléments à l'intérieur des crochets (**{}**). L'exemple suivant déclare, crée et initialise une variable devant contenir un tableau à deux dimensions avec des éléments du type de données **Short**, en spécifiant les limites supérieures et les valeurs. Notez les deux niveaux d'accolades dans la clause **New**.

```
Dim startingScores(,) As Short = New Short(1, 1) {{10, 10}, {10, 10}}
```

Après l'exécution de cette instruction, le tableau dans la variable `startingScores` contient quatre éléments initialisés. Si vous fournissez à la fois les limites supérieures et les valeurs, vous devez inclure une valeur pour chaque élément de l'index 0 jusqu'à la limite supérieure dans chaque dimension.

- ou -

- Dans la clause **New**, laissez les parenthèses vident à l'exception des virgules pour le nombre approprié de dimensions et fournissez les valeurs d'élément entre accolades (**{}**). L'exemple suivant déclare, crée et initialise une variable devant contenir un tableau à deux dimensions avec des éléments du type de données Single (Visual Basic), en spécifiant uniquement les valeurs d'éléments. Notez les deux niveaux d'accolades dans la clause **New**.

```
Dim diagonal(,) As Single = New Single(,) {{1, 0}, {0, 1}}
```

Après l'exécution de cette instruction, le tableau dans la variable `diagonal` contient quatre éléments initialisés.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau à plusieurs dimensions
Comment : initialiser une variable tableau
Comment : initialiser un tableau en escalier
Dépannage des tableaux

Concepts

Tableaux multidimensionnels dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Tableaux en escalier dans Visual Basic

Un tableau dont chaque élément est lui-même un tableau est appelé un *tableau de tableaux* ou un tableau *en escalier*. Notez qu'avoir des tableaux comme éléments n'est pas la même chose qu'un tableau *multidimensionnel* qui possède plusieurs index sur un tableau seul.

Signification d'« en escalier »

La structure des données de votre application est parfois à deux dimensions, mais pas rectangulaire. Vous pouvez par exemple posséder un tableau des mois dont chacun des éléments est un tableau des jours. Puisque des mois distincts possèdent un nombre de jours différent, les éléments ne constituent pas un tableau rectangulaire à deux dimensions. Dans ce cas, vous pouvez utiliser un tableau en escalier au lieu d'un tableau multidimensionnel.

Exemple

L'exemple suivant déclare une variable tableau pour stocker un tableau de tableaux avec les éléments du Double, type de données (Visual Basic). Chaque élément du tableau `sales` est lui-même un tableau qui représente un mois. Chaque tableau de mois stocke des valeurs pour chaque jour de ce mois.

```
Dim sales()() As Double = New Double(11)() {}
Dim month As Integer
Dim days As Integer
For month = 0 To 11
    days = DateTime.DaysInMonth(Year(Now), month + 1)
    sales(month) = New Double(days - 1) {}
Next month
```

La clause **New** dans la déclaration de `sales` attribue à la variable tableau la valeur d'un tableau de 12 éléments, chaque élément correspondant à un type **Double()**, un tableau d'éléments **Double**. La boucle **For** détermine ensuite le nombre de jours de chaque mois de cette année (**Year(Now)**) et définit l'élément correspondant de `sales` à un tableau **Double** de taille appropriée.

Dans l'exemple précédent, le tableau en escalier enregistre sept éléments (six dans une année bissextile) par rapport à un tableau à deux dimensions. Dans un cas plus extrême, les économies de mémoire pourraient être importantes.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau de tableaux
Comment : initialiser un tableau en escalier
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Dimensions du tableau dans Visual Basic
Tableaux multidimensionnels dans Visual Basic
Types de données tableau dans Visual Basic
Écriture d'un code conforme CLS

Autres ressources

Tableaux dans Visual Basic

Comment : créer un tableau de tableaux

Un tableau ayant comme éléments des tableaux porte le nom de *tableau de tableaux* ou de tableau *en escalier*. Comme avec un tableau unidimensionnel, vous le créez avec une clause New (Visual Basic) et l'assignez à la variable tableau. Cette opération peut s'effectuer dans le cadre de la déclaration de tableau ou dans une instruction d'assignation ultérieure.

Pour créer un tableau en escalier

1. Ajoutez autant de paires de parenthèses après le nom de variable qu'il n'existe de niveaux de tableaux imbriqués.
2. Ajoutez le même nombre de paires de parenthèses à la clause **New**. Il n'est pas nécessaire de spécifier plusieurs paires d'accolades (**{}**) si vous ne fournissez pas de valeurs d'éléments. L'exemple suivant déclare une variable devant contenir un tableau de tableaux avec des éléments de Byte, type de données (Visual Basic), crée le tableau et l'assigne à la variable.

   ```
   Dim ratings As Byte()() = New Byte()() {}
   ```

   Après l'exécution de cette instruction, la longueur du tableau dans la variable `ratings` est de 0.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : créer un tableau à plusieurs dimensions
Comment : créer un tableau avec des types d'élément mixtes
Comment : créer un tableau sans éléments
Comment : initialiser un tableau en escalier
Dépannage des tableaux

Concepts

Tableaux en escalier dans Visual Basic
Écriture d'un code conforme CLS

Autres ressources

Tableaux dans Visual Basic

Comment : initialiser un tableau en escalier

Lorsque vous initialisez une variable tableau en escalier, vous pouvez spécifier les longueurs de dimension uniquement pour le tableau de niveau supérieur. Vous pouvez effectuer cette opération de plusieurs manières :

Pour initialiser une variable tableau en escalier

- Dans la déclaration de variable tableau, spécifiez la limite supérieure d'index de niveau supérieur à l'intérieur de la première paire de parenthèses, séparée par des virgules. L'exemple suivant déclare et crée une variable pour contenir un tableau en escalier d'éléments **Byte** en spécifiant uniquement la limite supérieure de niveau supérieur.

```
Dim rainfall(11)() As Byte
```

Après l'exécution de cette instruction, le tableau dans la variable `rainfall` contient 12 éléments, chacun correspondant à un tableau vide d'éléments **Byte**.

- ou -

- Dans la clause **New**, spécifiez la limite supérieure d'index de niveau supérieur à l'intérieur de la première paire de parenthèses et ajoutez des accolades vides (**{}**). L'exemple suivant déclare et crée une variable pour contenir un tableau en escalier d'éléments **Short** en spécifiant uniquement la limite supérieure de niveau supérieur.

```
Dim snowfall()() As Short = New Short(11)() {}
```

Après l'exécution de cette instruction, le tableau dans la variable `snowfall` contient 12 éléments, chacun correspondant à un tableau vide d'éléments **Short**.

Remarque

Vous pouvez initialiser la limite supérieure d'index de niveau supérieur dans un seul emplacement. Si vous spécifiez un e limite supérieure dans les parenthèses qui suivent le nom de variable tableau, vous ne pouvez pas utiliser de clause **New**. Si vous spécifiez la limite supérieure dans les parenthèses dans la clause **New**, vous devez laisser vide les parent hèses qui suivent le nom de variable.

- ou -

- Dans la clause **New**, spécifiez la limite supérieure d'index de niveau supérieur à l'intérieur des parenthèses et fournissez les valeurs d'élément à l'intérieur des accolades (**{}**). L'exemple suivant déclare, crée, et initialise une variable destinée à contenir un tableau en escalier d'éléments **Char** en spécifiant la limite supérieure de niveau supérieur et les valeurs. Notez les clauses **New** imbriquées pour initialiser les tableaux de niveau inférieur.

```
Dim decodeValues()() As Char = New Char(1)() {New Char() {"a"c, "b"c}, New Char() {"p"
c, "q"c}}
```

Après l'exécution de cette instruction, le tableau dans la variable `decodeValues` contient deux éléments, chacun correspondant à un tableau **Char** de longueur 1 avec l'élément à index 0 contenant une valeur initialisée. Si vous fournissez à la fois la limite supérieure de niveau supérieur et les valeurs, vous devez inclure une valeur pour chaque élément de niveau supérieur de l'index 0 jusqu'à la limite supérieure.

- ou -

- Dans la clause **New**, omettez les parenthèses de niveau supérieur et fournissez les valeurs d'élément à l'intérieur des accolades (**{}**). L'exemple suivant déclare, crée et initialise une variable pour contenir un tableau en escalier d'éléments **Byte** en spécifiant uniquement les valeurs d'élément. Notez les deux niveaux d'accolades dans la clause **New**.

```
Dim firstValues()() As Byte = {New Byte() {2, 1}, New Byte() {3, 0}}
```

Après l'exécution de cette instruction, la longueur du tableau dans la variable `firstValues` est de 2 avec les éléments `firstValues(0)` et `firstValues(1)`. Chacun des éléments est initialisé avec un tableau **Byte** de deux éléments, le premier avec des valeurs d'élément 2 et 1 et le second avec des valeurs d'élément 3 et 0.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau de tableaux
Comment : initialiser une variable tableau
Comment : initialiser un tableau multidimensionnel
Dépannage des tableaux

Référence

New (Visual Basic)

Concepts

Tableaux en escalier dans Visual Basic
Écriture d'un code conforme CLS

Autres ressources

Tableaux dans Visual Basic

Types de données tableau dans Visual Basic

Il n'existe pas de type de données unique pour tous les tableaux. Au lieu de cela, le type de données d'un tableau est déterminé par les facteurs suivants :

- le fait qu'il s'agit d'un tableau ;
- le rang (nombre de dimensions) ;
- le type de données des éléments du tableau.

Par conséquent, deux variables tableau ne sont pas du même type de données, sauf si elles ont le même rang et que leurs éléments ont le même type de données.

Notez que les longueurs des dimensions n'influencent pas le type de données de tableau.

Exemples de tableaux

L'exemple suivant déclare quatre variables tableau qui ont différents types de données.

```
Dim firstArray(12, 8) As UInteger
Dim secondArray(12, 8, 3) As UInteger
Dim thirdArray(12, 8) As String
Dim fourthArray(5, 20) As UInteger
```

Suivant l'exécution des instructions précédentes, les relations de type de données suivantes s'appliquent aux variables tableau :

- Les variables et `firstArray` et `secondArray` ne sont pas du même type de données car elles ont des rangs différents.
- Les variables `firstArray` et `thirdArray` ne sont pas du même type de données car elles ont des types de données d'éléments différents.
- Les variables `firstArray` et `fourthArray` sont du même type de données et vous pouvez assigner l'une à l'autre.

Types de données de tableau en escalier

Les tableaux de tableaux, à savoir les tableaux qui contiennent d'autres tableaux comme éléments, portent également le nom de *tableaux en escalier* car les longueurs des tableaux d'éléments ne sont pas nécessairement égales. L'exemple suivant déclare deux variables de tableau en escalier qui ont des types de données différents.

```
Dim twoDimOfOneDim(,)() As Integer
Dim oneDimOfTwoDim()(,) As Integer
```

Le tableau dans `twoDimOfOneDim` est à deux dimensions et son type de données d'élément est **Integer()**, ou tableaux **Integer** unidimensionnels. Le tableau dans `oneDimOfTwoDim` est unidimensionnel et son type de données d'élément est **Integer(,)**, ou tableaux **Integer** à deux dimensions.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : déterminer le type de données d'un tableau
Dépannage des tableaux

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Dimensions du tableau dans Visual Basic
Tableaux multidimensionnels dans Visual Basic
Tableaux en escalier dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : déterminer le type de données d'un tableau

Gardez à l'esprit que le type de données d'un tableau n'est jamais le même que celui de ses éléments. Vous pouvez déterminer le type de données d'un tableau ou de ses éléments de plusieurs manières.

- Vous pouvez appeler la méthode System.Object.GetType sur la variable de façon à obtenir un objet Type pour le type de la variable à l'exécution. L'objet **Type** contient des informations complètes dans ses propriétés et méthodes.
- Vous pouvez passer la variable à TypeName, fonction (Visual Basic) pour obtenir **String** qui contient le nom du type à l'exécution.
- Vous pouvez passer la variable à VarType, fonction (Visual Basic) pour obtenir une valeur **VariantType** représentant la classification de type de la variable.

Pour déterminer le type de données d'un tableau

- Appelez **TypeName** sur le nom de tableau. Ne faites pas suivre le nom du tableau de parenthèses, car vous demandez le type du tableau lui-même.

```
Dim thisTwoDimArray(,) As Integer = New Integer(9, 9) {}
MsgBox("Type of thisTwoDimArray is " & TypeName(thisTwoDimArray))
```

  L'appel **MsgBox** affiche le message "Type of thisTwoDimArray is Integer(,)", qui vous indique à la fois le type d'élément et le nombre de dimensions. Il n'affiche pas les longueurs actuelles des dimensions, car elles ne font pas partie du type de données d'un tableau.

Pour déterminer le type de données d'un élément de tableau

- Sélectionnez un élément existant et appelez **TypeName** sur cet élément.

```
Dim thisTwoDimArray(,) As Integer = New Integer(9, 9) {}
MsgBox("Type of thisTwoDimArray(0, 0) is " & TypeName(thisTwoDimArray(0, 0)))
```

  L'appel **MsgBox** affiche le message "Type of thisTwoDimArray(0, 0) is Integer".

  Le type de données d'élément fait partie du type de données du tableau. Pour cette raison, vous ne pouvez pas modifier le type de données, même avec une instruction d'assignation ou une instruction **ReDim**.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Dépannage des tableaux

Référence

TypeName, fonction (Visual Basic)
VarType, fonction (Visual Basic)
VariantType, énumération

Concepts

Types de données tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Taille du tableau dans Visual Basic

La taille d'un tableau est le produit des longueurs de toutes ses dimensions. Elle représente le nombre total d'éléments contenus actuellement dans le tableau.

L'exemple suivant déclare un tableau à trois dimensions.

```
Dim prices(3, 4, 5) As Long
```

La taille globale du tableau dans la variable `prices` est *(3 + 1) x (4 + 1) x (5 + 1) = 120*.

Considérations relatives aux tailles de tableaux

Il convient de prendre en compte les facteurs suivants.

Longueur de dimension

L'index de chaque dimension est basé sur 0, ce qui signifie qu'il varie de 0 à sa limite supérieure. Par conséquent, la longueur d'une dimension donnée est égale à la limite supérieure déclarée pour cette dimension plus 1.

Limites de longueurs

La longueur de chaque dimension d'un tableau est limitée à la valeur maximale du type de données **Integer**, qui est *(2 ^ 31) - 1*. Toutefois, la taille totale d'un tableau est limitée également par la mémoire disponible sur votre système. Si vous essayez d'initialiser un tableau qui dépasse la quantité de mémoire vive disponible, le Common Language Runtime lève une exception OutOfMemoryException.

Taille et taille d'élément

La taille d'un tableau est indépendante du type de données de ses éléments. La taille représente toujours le nombre total d'éléments, et non le nombre d'octets de stockage qu'ils consomment.

Consommation de mémoire

Il n'est pas prudent de faire des suppositions quant à la manière dont un tableau est stocké en mémoire. Le stockage varie sur les plates-formes de largeurs de données différentes, par conséquent le même tableau peut consommer plus de mémoire sur un système 64 bits que sur un système 32 bits. Selon configuration du système lorsque vous initialisez un tableau, le Common Language Runtime (CLR) peut assigner le stockage de façon à comprimer le plus possible les éléments ou à tous les aligner sur des limites matérielles naturelles. De plus, un tableau requiert une charge de stockage pour ses informations de contrôle et cette charge augmente avec chaque dimension ajoutée.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : déterminer la taille d'un tableau
Comment : modifier la taille d'un tableau
Comment : déterminer la longueur d'une dimension d'un tableau
Dépannage des tableaux

Autres ressources

Tableaux dans Visual Basic

Comment : déterminer la taille d'un tableau

La taille globale d'un tableau est le produit des longueurs de toutes ses dimensions. La propriété Length du tableau retourne sa taille globale, qui représente le nombre total d'éléments actuellement contenus dans le tableau, et non le nombre d'octets qu'ils utilisent dans l'emplacement de stockage.

Pour déterminer la taille totale d'un tableau

- Lisez la propriété **Length** du tableau. Ne faites pas suivre le nom de tableau par des parenthèses.

```
Dim thisDoubleArray(,) As Char = New Char(4, 9) {}
MsgBox("Total length of thisDoubleArray is " & CStr(thisDoubleArray.Length))
```

  L'appel **MsgBox** affiche le message "Total length of thisDoubleArray is 50".

Vous pouvez rechercher la longueur de chaque dimension à l'aide de la méthode GetLength du tableau.

Vous pouvez modifier la longueur d'une dimension individuelle, ce qui modifie la taille globale. Toutefois, vous ne pouvez pas modifier le rang (le nombre de dimensions).

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : modifier la taille d'un tableau
Comment : déterminer la longueur d'une dimension d'un tableau
Dépannage des tableaux

Référence

Length

Concepts

Taille du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : modifier la taille d'un tableau

Vous redimensionnez une variable tableau en lui assignant un nouvel objet tableau. Vous pouvez utiliser une instruction d'assignation standard ou le ReDim, instruction (Visual Basic). Dans l'un et l'autre cas, le tableau d'origine est remplacé par un tableau complètement nouveau, et la variable tableau pointe vers le nouveau tableau.

Redimensionner un tableau vous aide à gérer la mémoire efficacement. Par exemple, vous pouvez démarrer avec un petit tableau, puis augmenter sa taille si vous avez besoin de plus d'éléments. Ou bien, vous pouvez démarrer avec un grand tableau, puis réduire sa taille lorsque vous n'en avez plus besoin dans son ensemble. Cette technique occupe la mémoire additionnelle uniquement lorsque vous en avez besoin.

Pour redimensionner une variable tableau à l'aide d'une instruction d'assignation standard

1. Créez le nouvel objet tableau en spécifiant les nouvelles longueurs de dimension.
2. Assignez le nouvel objet tableau à la variable tableau.

```
Dim thisArrayVariable() As Integer = New Integer(99) {}
thisArrayVariable = New Integer(49) {}
```

Pour redimensionner une variable tableau à l'aide de l'instruction ReDim

- Spécifiez les nouvelles longueurs de dimension pour la variable tableau dans l'instruction **ReDim**.

```
Dim thisArrayVariable() As Integer = New Integer(99) {}
ReDim thisArrayVariable(9)
```

Lorsque vous utilisez **ReDim** pour redimensionner un tableau, les valeurs existantes de ses éléments sont en principe perdues. Cependant, vous pouvez les conserver en ajoutant le mot clé **Preserve** à l'instruction **ReDim**.

Pour redimensionner une variable tableau en conservant des valeurs d'élément existantes

1. Spécifiez les nouvelles longueurs de dimension pour la variable tableau dans l'instruction **ReDim**.
2. Insérez le mot clé Preserve dans l'instruction **ReDim**. L'exemple suivant crée un nouveau tableau, initialise ses éléments à partir des éléments correspondants du tableau existant dans `arrayToIncrease` et assigne le nouveau tableau à la variable tableau `arrayToIncrease`.

```
Dim arrayToIncrease(9, 49)
ReDim Preserve arrayToIncrease(9, 199)
```

Si vous utilisez **Preserve** sur un tableau multidimensionnel, vous ne pouvez modifier que la dernière longueur de dimension. Si vous essayez de modifier une autre dimension, une exception ArrayTypeMismatchException se produit.

Si vous redimensionnez un grand tableau à l'aide du mot clé **Preserve**, n'oubliez pas que Visual Basic doit copier tous les éléments existants dans le nouveau tableau. Cela peut ralentir vos performances.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : déterminer la taille d'un tableau
Comment : déterminer la longueur d'une dimension d'un tableau
Comment : assigner un tableau à un autre tableau
Comment : remplacer un tableau par un autre tableau
Dépannage des tableaux

Concepts

Taille du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : déterminer la longueur d'une dimension d'un tableau

La méthode GetLength d'un tableau retourne la longueur de la dimension que vous spécifiez.

Pour déterminer la longueur d'une dimension d'un tableau

- Appelez **GetLength** sur le nom de tableau. Fournissez la dimension dont vous souhaitez obtenir la longueur comme argument de **GetLength**. Notez que l'argument de dimension est basé sur 0.

```
Dim sampleTripleArray(,,) As Short = New Short(2, 3, 4) {}
MsgBox("Dimension lengths of sampleTripleArray are " & CStr(sampleTripleArray.GetLengt
h(0)) _
    & ", " & CStr(sampleTripleArray.GetLength(1)) & ", " & CStr(sampleTripleArray.GetL
ength(2)))
```

  L'appel **MsgBox** affiche le message "Dimension lengths of sampleTripleArray are 3, 4, 5".

La valeur d'index la plus basse pour chaque dimension est toujours 0 et la méthode GetUpperBound retourne la valeur d'index la plus élevée d'une dimension. Pour chaque dimension, **GetLength** retourne une valeur supérieure de 1 par rapport à celle qui est retournée par **GetUpperBound**. Comme avec **GetLength**, la dimension que vous spécifiez pour **GetUpperBound** est basée sur 0.

Vous pouvez déterminer la longueur totale d'un tableau à partir de sa propriété Length.

Vous pouvez modifier la taille globale en modifiant la longueur d'une dimension individuelle. Toutefois, vous ne pouvez pas modifier le rang (le nombre de dimensions).

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : déterminer la taille d'un tableau
Comment : modifier la taille d'un tableau
Dépannage des tableaux

Concepts

Taille du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Utilisation des tableaux et les éléments de tableau

Cette section aborde le tri de tableaux, le placement ou l'obtention d'une valeur de tableau, ainsi que d'autres actions sur les tableaux et les éléments qu'ils contiennent.

Dans cette section

Comment : placer une valeur dans un tableau

Montre comment stocker des valeurs dans des éléments de tableau.

Comment : obtenir une valeur d'un tableau

Décrit comment récupérer des valeurs d'éléments de tableau.

Comment : localiser un élément dans un tableau dans Visual Basic

Explique comment rechercher un élément particulier dans un tableau.

Comment : inverser le contenu d'un tableau dans Visual Basic

Explique comment inverser l'ordre des éléments d'un tableau.

Comment : trier un tableau dans Visual Basic

Montre comment trier alphabétiquement les éléments d'un tableau.

Comment : assigner un tableau à un autre tableau

Décrit les règles et les étapes à suivre pour assigner un tableau à une autre variable de tableau.

Comment : remplacer un tableau par un autre tableau

Décrit les modifications possibles et comment les effectuer.

Comment : passer un tableau à une procédure ou propriété

Explique comment passer un tableau en tant qu'argument à une procédure ou à une propriété.

Comment : retourner un tableau à partir d'une procédure ou propriété

Décrit comment retourner un tableau au code qui appelle une procédure ou une propriété.

Référence

Dim, instruction (Visual Basic)

Déclare et alloue un espace de stockage pour une ou plusieurs variables.

System.Array

Fournit des méthodes de création, de manipulation, de recherche et de tri de tableaux ; sert donc de classe de base pour tous les tableaux du Common Language Runtime.

Rubriques connexes

Dépannage des tableaux

Aborde certains problèmes courants qui surviennent lors de l'utilisation des tableaux.

Tableaux dans Visual Basic

Explique comment rendre votre code plus condensé et puissant en déclarant et en utilisant des tableaux.

Comment : placer une valeur dans un tableau

Vous pouvez stocker l'une des valeurs d'un tableau en utilisant le nom du tableau et les index appropriés pour spécifier l'élément individuel.

Pour placer une valeur dans un élément de tableau

1. À gauche du signe égal (**=**), spécifiez le nom du tableau suivi de parenthèses.
2. Dans les parenthèses, incluez une expression pour chaque index correspondant à l'élément que vous souhaitez stocker. Un index est nécessaire pour chaque dimension de tableau. L'exemple suivant montre quelques instructions qui stockent des valeurs dans des tableaux.

   ```
   Dim numbers() As Integer

   Dim matrix(,) As Double

   numbers(i + 1) = 0

   matrix(3, j * 2) = j
   ```

   Pour chaque dimension de tableau, la méthode GetUpperBound retourne la valeur la plus élevée que l'index peut comporter. La valeur d'index la plus basse est toujours 0.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : obtenir une valeur d'un tableau
Comment : localiser un élément dans un tableau dans Visual Basic
Comment : inverser le contenu d'un tableau dans Visual Basic
Comment : trier un tableau dans Visual Basic
Dépannage des tableaux

Concepts

Dimensions du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : obtenir une valeur d'un tableau

Vous pouvez obtenir l'une des valeurs d'un tableau en utilisant le nom du tableau et des index appropriés pour spécifier l'élément individuel.

Pour obtenir une valeur d'un élément de tableau

1. Au sein d'une expression, spécifiez le nom du tableau suivi de parenthèses.
2. Dans les parenthèses, incluez une expression pour chaque index correspondant à l'élément que vous souhaitez obtenir. Vous avez besoin d'un index pour chaque dimension de tableau. L'exemple suivant montre des instructions qui obtiennent des valeurs à partir de tableaux.

```
Dim sortedValues(), rawValues(), estimates(,,) As Double
lowestValue = sortedValues(0)
wTotal += (rawValues(v) ^ 2)
firstGuess = estimates(i, j, k)
```

Pour chaque dimension de tableau, la méthode GetUpperBound retourne la valeur la plus élevée que l'index peut comporter. La valeur d'index la plus basse est toujours 0.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : placer une valeur dans un tableau
Comment : localiser un élément dans un tableau dans Visual Basic
Comment : inverser le contenu d'un tableau dans Visual Basic
Comment : trier un tableau dans Visual Basic
Dépannage des tableaux

Concepts

Dimensions du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : localiser un élément dans un tableau dans Visual Basic

Cet exemple déclare un tableau d'objets **String** nommé `zooAnimals`, le remplit, localise l'élément « turtle » et affiche son emplacement.

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

```
Public Sub findAnimal()
    Dim zooAnimals(2) As String
    zooAnimals(0) = "lion"
    zooAnimals(1) = "turtle"
    zooAnimals(2) = "ostrich"
    Dim turtleIndex As Integer
    turtleIndex = (Array.IndexOf(zooAnimals,"turtle"))
    MsgBox("The turtle is element " & turtleIndex)
End Sub
```

Compilation du code

Cet exemple nécessite :

- Accès à Mscorlib.dll et à l'espace de noms System.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- Le tableau est vide (classe ArgumentNullException).
- Le tableau est multidimensionnel (classe RankException).
- Un ou plusieurs éléments du tableau n'implémentent pas l'interface IComparable (classe InvalidOperationException).

Voir aussi

Tâches

Comment : placer une valeur dans un tableau
Comment : obtenir une valeur d'un tableau
Comment : inverser le contenu d'un tableau dans Visual Basic
Comment : trier un tableau dans Visual Basic
Dépannage des tableaux

Référence

System.Array.IndexOf

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : inverser le contenu d'un tableau dans Visual Basic

Cet exemple déclare un tableau d'objets **String** nommé `zooAnimals`, le remplit et inverse son contenu.

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

```
Public Sub reverseAnimals()
    Dim zooAnimals(2) As String
    zooAnimals(0) = "lion"
    zooAnimals(1) = "turtle"
    zooAnimals(2) = "ostrich"
    Array.Reverse(zooAnimals)
End Sub
```

Compilation du code

Cet exemple nécessite :

- Accédez à Mscorlib.dll et à l'espace de noms System.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception.

- Le tableau est vide (classe ArgumentNullException).
- Le tableau est multidimensionnel (classe RankException).
- Un ou plusieurs éléments du tableau n'implémentent pas l'interface IComparable (classe InvalidOperationException).

Voir aussi

Tâches

Comment : placer une valeur dans un tableau
Comment : obtenir une valeur d'un tableau
Comment : localiser un élément dans un tableau dans Visual Basic
Comment : trier un tableau dans Visual Basic
Dépannage des tableaux

Référence

System.Array.Reverse

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : trier un tableau dans Visual Basic

Cet exemple déclare un tableau d'objets **String** nommé `zooAnimals`, le remplit et le trie par ordre alphabétique.

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

```
Private Sub sortAnimals()
    Dim zooAnimals(2) As String
    zooAnimals(0) = "lion"
    zooAnimals(1) = "turtle"
    zooAnimals(2) = "ostrich"
    Array.Sort(zooAnimals)
End Sub
```

Compilation du code

Cet exemple nécessite :

- Accédez à Mscorlib.dll et à l'espace de noms System.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception.

- Le tableau est vide (classe ArgumentNullException).
- Le tableau est multidimensionnel (classe RankException).
- Un ou plusieurs éléments du tableau n'implémentent pas l'interface IComparable (classe InvalidOperationException).

Voir aussi

Tâches

Comment : placer une valeur dans un tableau
Comment : obtenir une valeur d'un tableau
Comment : localiser un élément dans un tableau dans Visual Basic
Comment : inverser le contenu d'un tableau dans Visual Basic
Dépannage des tableaux

Référence

System.Array.Sort

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : assigner un tableau à un autre tableau

Étant donné que les tableaux sont des objets, ils peuvent être utilisés dans des instructions d'assignation comme les autres types d'objets. Une variable tableau maintient un pointeur vers les données constituant les éléments de tableau et les informations relatives au rang et à la longueur. Une assignation ne copie que ce pointeur.

Pour assigner un tableau à un autre tableau

1. Vérifiez que les deux tableaux ont le même rang (nombre de dimensions) et des types de données d'élément compatibles.
2. Utilisez une instruction d'assignation standard pour assigner le tableau source au tableau de destination. Ne faites pas suivre les noms du tableau de parenthèses.

```
Dim formArray() As System.Windows.Forms.Form
Dim controlArray() As System.Windows.Forms.Control
controlArray = formArray
```

Les règles suivantes s'appliquent lors de l'assignation d'un tableau à un autre :

- **Rangs identiques.** Le rang (nombre de dimensions) du tableau de destination doit être identique à celui du tableau source.

  Si les rangs des deux tableaux sont identiques, les dimensions n'ont pas besoin de l'être. Le nombre d'éléments dans une dimension donnée peut varier lors de l'assignation.

- **Types d'éléments.** Les deux tableaux doivent avoir des éléments de *type référence* ou de *type valeur*. Pour plus d'informations, consultez Types valeur et types référence.

  - Si les deux tableaux ont des éléments de type valeur, les types de données d'élément doivent être exactement identiques. La seule exception est que vous pouvez assigner un tableau d'éléments **Enum** à un tableau du type de base de **Enum**.
  - Si les deux tableaux ont des éléments de type référence, le type d'élément source doit dériver du type d'élément de destination. Si c'est le cas, les deux tableaux ont la même relation d'héritage que leurs éléments. Cette situation s'appelle la *covariance de tableau*.

Le compilateur signale une erreur si les règles énoncées ci-dessus ne sont pas respectées ; par exemple, si les types de données ne sont pas compatibles ou si les rangs ne sont pas identiques. Vous pouvez ajouter une gestion des erreurs à votre code pour vous assurer que les tableaux sont compatibles avant d'effectuer une assignation. Vous pouvez également faire appel au mot clé TryCast si vous souhaitez éviter de lever une exception.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : modifier la taille d'un tableau
Comment : remplacer un tableau par un autre tableau
Comment : passer un tableau à une procédure ou propriété
Comment : retourner un tableau à partir d'une procédure ou propriété
Dépannage des tableaux

Référence

Enum, instruction (Visual Basic)

Concepts

Conversion des tableaux

Autres ressources

Tableaux dans Visual Basic

Comment : remplacer un tableau par un autre tableau

Il est important de faire la différence entre un *objet tableau* et une *variable tableau*. Une variable tableau contient un pointeur vers un objet tableau qui contient les éléments de tableau et les informations de rang et de longueur.

- Une fois que vous créez un objet tableau, vous ne pouvez pas modifier son rang (nombre de dimensions), ses longueurs de dimension ni le type de données de ses éléments. Vous pouvez modifier uniquement le contenu de ses éléments.
- Une fois que vous déclarez une variable tableau, vous ne pouvez pas modifier son rang ni son type de données d'élément. Toutefois, vous pouvez assigner une suite de différents objets tableau à la variable pendant sa durée de vie. Ces objets tableau peuvent avoir des longueurs de dimension différentes.

Pour modifier une variable tableau pour qu'elle pointe vers un autre objet tableau

- Utilisez une instruction d'assignation standard pour assigner le tableau source au tableau de destination.

```
Dim array1(4), array2(19) As String
array2 = array1
```

Vous pouvez modifier une variable tableau pour qu'elle pointe vers un objet tableau avec des longueurs de dimension différentes, mais vous ne pouvez pas la modifier pour qu'elle pointe vers un objet tableau d'un type de données différent. Cela signifie que le rang et le type de données d'élément doivent rester les mêmes puisqu'ils font partie intégrante du type de données de la variable tableau.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : assigner un tableau à un autre tableau
Comment : passer un tableau à une procédure ou propriété
Comment : retourner un tableau à partir d'une procédure ou propriété
Comment : modifier la taille d'un tableau
Dépannage des tableaux

Autres ressources

Tableaux dans Visual Basic

Comment : passer un tableau à une procédure ou propriété

Le passage d'un tableau s'effectue comme pour toute autre variable. Vous fournissez le nom de la variable tableau dans l'argument approprié lorsque vous appelez la procédure ou accédez à la propriété.

Pour passer un tableau à une procédure

1. Assurez-vous que l'un des paramètres de procédure spécifie un tableau avec les mêmes rang (nombre de dimensions) et type de données d'élément.
2. Fournissez la variable tableau à l'emplacement correspondant dans la liste d'arguments. Ne faites pas suivre le nom de tableau par des parenthèses.

```
Public Function findLargest(ByVal numbers() As Double) As Double
    ' Insert code to calculate and return largest number.
End Function
Dim testNumbers() As Double = New Double() {5.0, 3.7, 1.2, 7.6}
Dim largestNumber As Double = findLargest(testNumbers)
```

Pour passer un tableau à une propriété

1. Assurez-vous que l'un des paramètres de propriété spécifie un tableau avec les mêmes rang (nombre de dimensions) et type de données d'élément.
2. Fournissez la variable tableau à l'emplacement correspondant dans la liste d'arguments. Ne faites pas suivre le nom de tableau par des parenthèses.

```
Public Property bestMatch(ByVal formattedStrings() As String) As Double
    ' Insert Get and Set procedures for number best matching strings.
End Property
Dim testStrings() As String = New String() {}
Dim formattedNumber As Double = bestMatch(testStrings)
```

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : assigner un tableau à un autre tableau
Comment : remplacer un tableau par un autre tableau
Comment : retourner un tableau à partir d'une procédure ou propriété
Dépannage des tableaux

Autres ressources

Tableaux dans Visual Basic

Comment : retourner un tableau à partir d'une procédure ou propriété

Vous retournez un tableau de la même manière que tout autre type de données. Vous fournissez le type tableau comme type de retour de la procédure ou propriété.

Pour retourner un tableau à partir d'une procédure Function

1. Spécifiez le type tableau (rang et type de données d'élément) comme type de retour dans l'instruction Function (Visual Basic).
2. Dans la procédure, déclarez une variable tableau locale avec les mêmes rang et type de données d'élément.
3. Incluez cette variable tableau locale dans Return, instruction (Visual Basic). Ne faites pas suivre le nom de tableau par des parenthèses.

```
Public Function splitNumber(ByVal number As Double) As Char()
    Dim characters() As Char
    ' Insert code to split number into characters.
    Return characters
End Function
Dim piCharacters() As Char = splitNumber(3.14159265)
```

Pour retourner un tableau à partir d'une propriété

1. Spécifiez le type tableau (rang et type de données d'élément) comme type de propriété dans l'instruction Property.
2. Dans la procédure Get de la propriété, ou à un emplacement accessible à la procédure **Get**, déclarez une variable tableau locale avec les mêmes rang et type de données d'élément.
3. Incluez cette variable tableau locale dans l'instruction **Return**. Ne faites pas suivre le nom de tableau par des parenthèses.

```
Private nameList() As String
Public Property stationNames As String()
    Get
        Return nameList
    End Get
    Set(ByVal Value As String())
        ' Insert code to store nameList values.
    End Set
End Property
Dim listOfNames() As String = stationNames
```

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Comment : assigner un tableau à un autre tableau
Comment : remplacer un tableau par un autre tableau
Comment : passer un tableau à une procédure ou propriété
Dépannage des tableaux

Autres ressources

Tableaux dans Visual Basic

Collections en tant qu'alternative aux tableaux

Bien que les collections soient souvent utilisées pour fonctionner avec le type de données Object, vous pouvez les utiliser avec tout autre type de données. Parfois, il peut être plus efficace d'enregistrer des éléments dans une collection plutôt que dans un tableau.

Si vous devez modifier la taille d'un tableau, vous devez utiliser l'ReDim, instruction (Visual Basic). Dans ce cas, Visual Basic crée un tableau et libère le tableau précédent pour qu'il soit éliminé. Cela nécessite un temps d'exécution. Par conséquent, si le nombre d'éléments avec lesquels vous travaillez change fréquemment ou si vous ne pouvez pas prévoir le nombre maximal d'éléments dont vous aurez besoin, vous pourriez obtenir des performances supérieures en utilisant une collection.

Une collection, qui n'est pas obligée de créer de nouvel objet ni de copier d'éléments existants, peut gérer le redimensionnement en moins de temps d'exécution qu'un tableau, qui doit utiliser **ReDim**. Mais si la taille ne change pas ou change rarement, il est probable qu'un tableau constitue une solution plus efficace. Comme toujours, les performances dépendent beaucoup de l'application. Il vaut souvent la peine d'essayer à la fois un tableau et une collection.

Collections spécialisées

Le .NET Framework fournit également diverses classes, interfaces et structures pour les collections générales et spéciales. Les espaces de noms System.Collections et System.Collections.Specialized contiennent des définitions et des implémentations qui incluent des dictionnaires, des listes, des files d'attente et des piles. L'espace de noms System.Collections.Generic fournit une grande partie de ceux-ci dans des versions génériques, qui prennent un ou plusieurs arguments de type.

Si votre collection doit contenir des éléments d'un seul type de données spécifique, une collection générique offre l'avantage de mettre en vigueur la *sécurité de type*. Pour plus d'informations sur les types génériques, consultez Types génériques en Visual Basic.

Exemple

L'exemple suivant utilise la classe générique System.Collections.Generic.List du .NET Framework pour créer une collection de listes de structures `customer`.

```
' Define the structure for a customer.
Public Structure customer
    Public name As String
    ' Insert code for other members of customer structure.
End Structure
' Create a module-level collection that can hold 200 elements.
Public custFile As New List(Of customer)(200)
' Add a specified customer to the collection.
Private Sub addNewCustomer(ByVal newCust As customer)
    ' Insert code to perform validity check on newCust.
    custFile.Add(newCust)
End Sub
' Display the list of customers in the Debug window.
Private Sub printCustomers()
    For Each cust As customer In custFile
        Debug.WriteLine(cust)
    Next cust
End Sub
```

La déclaration de la collection `custFile` spécifie qu'elle peut contenir uniquement des éléments de type `customer`. Elle prévoit également une capacité initiale de 200 éléments. La procédure `addNewCustomer` vérifie que le nouvel élément est valide, puis elle l'ajoute à la collection. La procédure `printCustomers` utilise une boucle **For Each** pour parcourir la collection et afficher ses éléments.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : initialiser une variable tableau
Dépannage des tableaux

Référence

ReDim, instruction (Visual Basic)

Concepts

Collections en Visual Basic

Types génériques en Visual Basic

Autres ressources

Tableaux dans Visual Basic

Dépannage des tableaux

Cette page répertorie quelques problèmes courants qui peuvent se produire lors de l'utilisation des tableaux.

Déclaration d'erreurs de compilation et initialisation d'un tableau

Les erreurs de compilation peuvent être dues à une mauvaise compréhension des règles de déclaration, de création et d'initialisation des tableaux. Les causes d'erreurs les plus courantes sont les suivantes :

- fournir une clause New (Visual Basic) après avoir spécifié les longueurs de dimension dans la déclaration des variables du tableau. Les lignes de code suivantes affichent des déclarations non valides de ce type ;

```
Dim INVALIDsingleDimByteArray(2) As Byte = New Byte()

Dim INVALIDtwoDimShortArray(1, 1) As Short = New Short(,)

Dim INVALIDjaggedByteArray(1)() As Byte = New Byte()()
```

- spécifier des longueurs de dimension supérieures au tableau de niveau supérieur d'un tableau en escalier. La ligne de code suivante affiche une déclaration non valide de ce type ;

```
Dim INVALIDjaggedByteArray(1)(1) As Byte
```

- omettre le mot clé **New** lors de la spécification des valeurs d'éléments. La ligne de code suivante affiche une déclaration non valide de ce type ;

```
Dim INVALIDoneDimShortArray() As Short = Short() {0, 1, 2, 3}
```

- fournir une clause **New** sans accolades (**{}**). Les lignes de code suivantes affichent des déclarations non valides de ce type.

```
Dim INVALIDsingleDimByteArray() As Byte = New Byte()

Dim INVALIDsingleDimByteArray() As Byte = New Byte(2)

Dim INVALIDtwoDimShortArray(,) As Short = New Short(,)

Dim INVALIDtwoDimShortArray(,) As Short = New Short(1, 1)
```

Accès à un tableau hors limites

Le processus d'initialisation d'un tableau assigne une limite supérieure et une limite inférieure à chaque dimension. Chaque accès à un élément du tableau doit spécifier un index ou un indice valide pour chaque dimension. Si un index est en dessous de sa limite inférieure ou au-dessus de sa limite supérieure, une d'exception IndexOutOfRangeException se produit. Le compilateur ne peut pas détecter une telle erreur, donc une erreur se produit au moment de l'exécution.

Détermination des limites

Si un autre composant passe un tableau à votre code, par exemple en tant qu'argument de procédure, vous ne connaissez pas la taille de ce tableau ou les longueurs de ses dimensions. Vous devez toujours déterminer la limite supérieure de chaque dimension d'un tableau avant d'essayer d'accéder aux éléments. Si le tableau a été créé par d'autres moyens qu'une clause **New** Visual Basic, la limite inférieure peut avoir une valeur différente de 0, et elle est également la valeur la plus sûre pour déterminer cette limite inférieure.

Spécification de la dimension

Lorsque vous déterminez les limites d'un tableau multidimensionnel, veillez à bien spécifier la dimension. Les paramètres *dimension* des méthodes GetLowerBound et GetUpperBound sont de base 0, tandis que les paramètres *Rank* de LBound, fonction (Visual Basic) et UBound, fonction (Visual Basic) de Visual Basic sont de base 1.

Voir aussi

Tâches

Comment : déclarer une variable tableau
Comment : créer un tableau
Comment : créer un tableau à plusieurs dimensions
Comment : créer un tableau de tableaux
Comment : créer un tableau avec des types d'élément mixtes
Comment : créer un tableau sans éléments

Comment : initialiser une variable tableau
Comment : initialiser un tableau multidimensionnel
Comment : initialiser un tableau en escalier

Autres ressources

Tableaux dans Visual Basic

Objets en Visual Basic

Lorsque vous créez une application en Visual Basic, vous utilisez constamment des objets. Vous pouvez utiliser des objets fournis par Visual Basic, tels que les contrôles, les formulaires et les objets d'accès aux données. Vous pouvez également utiliser des objets d'autres applications contenues dans votre application Visual Basic. Vous pouvez même créer vos propres objets et définir des propriétés ainsi que des méthodes supplémentaires pour ces objets. Les objets se comportent comme des blocs de construction préfabriqués : vous pouvez écrire un fragment de code une seule fois et le réutiliser continuellement.

Les objets sont traités en détail dans les rubriques suivantes :

Dans cette section

Objets et classes

Présente une vue d'ensemble des objets et des classes.

Comment : créer un objet

Explique comment créer un objet afin que vous puissiez utiliser ses membres.

Comment : réutiliser un composant existant

Explique comment utiliser les fonctionnalités qui ont déjà été écrites et déboguées.

Comment : définir une classe qui utilise les membres d'une classe existante

Explique comment utiliser les propriétés et les méthodes d'une classe de base dans votre classe dérivée.

Relations entre les objets

Présente les hiérarchies d'objets, les conteneurs et les collections.

Membres objets

Récapitule les champs, propriétés, méthodes et événements exposés par des objets.

Comment : accéder aux membres partagés et non partagés d'un objet

Explique comment lire, écrire ou appeler un membre d'un objet que vous avez créé.

Rubriques connexes

Introduction aux objets dans Visual Basic

Présente une vue d'ensemble de l'utilisation des objets en Visual Basic.

Création et utilisation d'objets

Montre comment créer et utiliser des instances de classes.

Objets et classes

Un *objet* est une combinaison de code et de données qui peut être traitée comme une entité unique. Un objet peut être une partie d'une application (telle qu'un contrôle ou un formulaire) ou une application dans son intégralité.

Réutilisation de code

Les objets vous permettent de déclarer une seule fois des variables et des procédures, puis de les réutiliser quand bon vous semble. Par exemple, pour ajouter un vérificateur d'orthographe à une application, vous pouvez définir toutes les variables et les fonctions de support pour assurer une vérification orthographique. Si vous créez votre vérificateur d'orthographe sous forme de classe, vous pouvez ensuite le réutiliser dans d'autres applications en ajoutant une référence à l'assembly compilé. Vous pouvez même utiliser une classe de vérificateur d'orthographe développée par quelqu'un d'autre pour vous éviter tout travail superflu.

Classes

Chaque objet dans Visual Basic est défini par une *classe*. Une classe décrit les variables, propriétés, procédures et événements d'un objet. Les objets sont des instances de classes ; vous pouvez créer autant d'objets que nécessaire dès que vous avez défini une classe.

Pour comprendre la relation existant entre un objet et sa classe, pensez à celle existant entre un moule à gâteau et des gâteaux. Le moule à gâteau représente la classe. Il définit les caractéristiques de chaque gâteau (sa taille et sa forme, par exemple). La classe permet de créer des objets. Les objets peuvent être comparés aux gâteaux.

Deux exemples dans Visual Basic peuvent aider à illustrer la relation entre les classes et les objets.

- Les contrôles de la **Boîte à outils** dans Visual Basic représentent des classes. Lorsque vous faites glisser un contrôle de la **Boîte à outils** vers un formulaire, vous créez un objet, c'est-à-dire une instance d'une classe.
- Le formulaire que vous utilisez au moment de la création est une classe. Au moment de l'exécution, Visual Basic crée une instance de la classe du formulaire, c'est-à-dire un objet.

Instances multiples

Les objets récemment créés à partir d'une classe sont souvent identiques les uns aux autres. Cependant, une fois qu'ils existent en tant qu'objets individuels, leurs variables et propriétés peuvent être modifiées indépendamment des autres instances. Par exemple, si vous ajoutez trois cases à cocher à un formulaire, chacun de ces objets est une instance de la classe CheckBox. Les objets **CheckBox** individuels partagent un ensemble commun de caractéristiques et de fonctionnalités (propriétés, variables, procédures et événements) définies par la classe. Cependant, chacun possède un nom qui lui est propre, il peut être activé ou désactivé séparément et placé à un autre emplacement du formulaire.

Voir aussi

Concepts

Relations entre les objets

Introduction aux objets dans Visual Basic

Autres ressources

Création et utilisation d'objets

Présentation des classes

Comment : créer un objet

Un objet est une instance d'une classe. Pour pouvoir utiliser les membres de la classe, vous devez d'abord créer un objet à partir de cette classe.

Pour créer un objet à partir d'une classe

1. Déterminez à partir de quelle classe vous souhaitez créer un objet.
2. Écrivez une Dim, instruction (Visual Basic) pour créer une variable à laquelle vous pouvez assigner une instance de classe. La variable doit être du type de la classe souhaitée.

```
Dim nextCustomer As customer
```

3. Ajoutez le mot clé New (Visual Basic) pour initialiser la variable à une nouvelle instance de la classe.

```
Dim nextCustomer As New customer
```

4. Vous pouvez maintenant accéder aux membres de la classe par le biais de la variable objet.

```
nextCustomer.accountNumber = lastAccountNumber + 1
```

Programmation fiable

Si possible, vous devez déclarer la variable comme étant du type classe que vous envisagez de lui assigner. On appelle cela une *liaison anticipée*. Si vous ne connaissez pas le type classe au moment de la compilation, vous pouvez appeler la *liaison tardive* en déclarant la variable comme étant de type de données Object. Toutefois, la liaison tardive peut ralentir les performances et limiter l'accès aux membres de l'objet à l'exécution. Pour plus d'informations, consultez Déclaration des variables objets.

Voir aussi

Tâches

Comment : réutiliser un composant existant
Comment : définir une classe qui utilise les membres d'une classe existante
Comment : accéder aux membres partagés et non partagés d'un objet

Concepts

Objets et classes

Comment : réutiliser un composant existant

Si un composant existe déjà, qu'il est débogué et qu'il fonctionne, il vaut mieux l'utiliser dans votre code que de développer un autre composant avec les mêmes fonctionnalités. Un tel composant est généralement exposé en tant que classe. Pour le réutiliser, vous devez créer un objet à partir de cette classe.

Exemple

Le .NET Framework fournit de nombreux exemples de composants disponibles. L'un de ces composants est la classe TimeZone dans l'espace de noms System. **TimeZone** fournit des membres qui vous permettent de récupérer des informations sur le fuseau horaire de votre système informatique.

```
Public Sub examineTimeZone()
    Dim tz As System.TimeZone = System.TimeZone.CurrentTimeZone
    Dim s As String = "Current time zone is "
    s &= CStr(tz.GetUtcOffset(Now).Hours) & " hours and "
    s &= CStr(tz.GetUtcOffset(Now).Minutes) & " minutes "
    s &= "different from UTC (coordinated universal time)"
    s &= vbCrLf & "and is currently "
    If tz.IsDaylightSavingTime(Now) = False Then s &= "not "
    s &= "on ""summer time""."
    MsgBox(s)
End Sub
```

Le premier Dim, instruction (Visual Basic) déclare une variable objet de type **TimeZone** et lui assigne un objet **TimeZone** retourné par la propriété CurrentTimeZone.

Voir aussi

Tâches

Comment : créer un objet

Comment : définir une classe qui utilise les membres d'une classe existante

Comment : accéder aux membres partagés et non partagés d'un objet

Concepts

Objets et classes

Comment : définir une classe qui utilise les membres d'une classe existante

Vous pouvez utiliser des membres d'une classe existante dans une autre classe qui en dérive.

Dans l'exemple suivant, supposez que vous souhaitez définir un genre spécial de Button qui agit comme un **Button** normal, mais qui expose également une méthode qui inverse les couleurs de premier plan et d'arrière-plan.

Pour définir une classe qui utilise les membres d'une classe existante

1. Utilisez une Class, instruction (Visual Basic) pour définir une classe à partir de laquelle créer l'objet dont vous avez besoin.

   ```
   Public Class reversibleButton
   ```

   Assurez-vous qu'une instruction **End Class** suit la dernière ligne de code dans votre classe. Par défaut, l'environnement de développement intégré (IDE) génère automatiquement une instruction **End Class** lorsque vous entrez une instruction **Class**.

2. Faites suivre immédiatement l'instruction **Class** d'une instruction Inherits. Spécifiez la classe à partir de laquelle dérive votre nouvelle classe.

   ```
   Inherits System.Windows.Forms.Button
   ```

   Votre nouvelle classe hérite de tous les membres définis par la classe de base.

3. Ajoutez le code pour les membres supplémentaires exposés par votre classe dérivée. Par exemple, vous pourriez ajouter une méthode `reverseColors` et votre classe dérivée pourrait ressembler à ce qui suit :

   ```
   Public Class reversibleButton
       Inherits System.Windows.Forms.Button
       Public Sub reverseColors()
           Dim saveColor As System.Drawing.Color = Me.BackColor
           Me.BackColor = Me.ForeColor
           Me.ForeColor = saveColor
       End Sub
   End Class
   ```

   Si vous créez un objet à partir de la classe `reversibleButton`, il peut accéder à tous les membres de la classe **Button**, ainsi qu'à la méthode `reverseColors` et à tout autre nouveau membre que vous définissez sur `reversibleButton`.

Compilation du code

Assurez-vous que le compilateur peut accéder à la classe à partir de laquelle vous envisagez de dériver votre nouvelle classe. Cela peut signifier qualifier complètement son nom, comme dans l'exemple précédent, ou identifier son espace de noms dans une instruction Imports. Si la classe est dans un projet différent, vous devrez peut-être ajouter une référence à ce projet. Pour plus d'informations, consultez Références aux espaces de noms et aux composants.

Voir aussi

Tâches

Comment : créer un objet
Comment : réutiliser un composant existant
Comment : accéder aux membres partagés et non partagés d'un objet
Comment : créer des classes dérivées

Concepts

Objets et classes
Éléments fondamentaux de l'héritage

Relations entre les objets

Les objets peuvent être reliés les uns aux autres de diverses manières. Les principales sortes de relations sont les relations *hiérarchiques* et les relations *contenant-contenu*.

Relation hiérarchique

Lorsque des classes sont dérivées de classes plus fondamentales, on dit qu'elles ont une *relation hiérarchique*. Les hiérarchies de classe sont utiles lors d'une description d'éléments qui représentent un sous-type d'une classe plus générale. Par exemple, dans l'espace de noms System.Windows.Forms, les classe Label et TextBox dérivent toutes deux de la classe Control. Dans la mesure où les classes dérivées héritent des membres de la classe dont elles sont issues, elles peuvent devenir plus complexes au fur et à mesure que vous avancez dans une hiérarchie de classe.

Relation contenant-contenu

Les objets peuvent également entretenir une *relation contenant-contenu*. Les objets conteneurs encapsulent d'autres objets. Par exemple, l'objet OperatingSystem contient logiquement un objet Version, qu'il retourne par le biais de sa propriété Version. Notez que l'objet conteneur ne contient pas physiquement d'autre objet.

Les collections

Les *collections* représentent un type particulier de relation contenant-contenu d'un objet. Les collections sont des groupes d'objets semblables qui peuvent être énumérés. Visual Basic prend en charge une syntaxe spécifique dans l'For Each...Next, instruction (Visual Basic) qui vous permet de parcourir les éléments d'une collection. En outre, les collections vous permettent souvent d'utiliser une Item, propriété (objet Collection) pour extraire des éléments par leur index ou en les associant à une chaîne unique. Les collections peuvent être plus simples à utiliser que les tableaux car elles vous permettent d'ajouter ou d'enlever des éléments sans utiliser d'index. Grâce à leur simplicité d'utilisation, elles sont souvent utilisées pour stocker des formulaires et des contrôles.

Voir aussi

Concepts

Membres objets
Collections en Visual Basic
Objets issus de Visual Basic et d'autres sources

Autres ressources

Design d'une hiérarchie d'héritage
Création et utilisation d'objets

Membres objets

Les objets sont les unités de base de la programmation orientée objet. Un objet est un élément d'une application, représentant une *instance* d'une classe. Les champs, les propriétés, les méthodes et les événements sont les blocs de construction des objets et constituent leurs *membres*.

Objects

Un objet représente une instance d'une classe, telle que Form ou Label. Vous devez créer un objet avant de pouvoir accéder à ses membres non partagés. Pour cela, vous devez utiliser le mot clé New pour spécifier la classe à partir de laquelle créer l'objet, puis assigner le nouvel objet à une variable objet.

```
Dim warningLabel As New System.Windows.Forms.Label
```

Pour plus d'informations, consultez Comment : créer un objet.

Accès au membre

Pour accéder à un membre d'un objet, vous devez spécifier le nom de la variable objet suivi d'un point (**.**) et du nom du membre. L'exemple suivant définit la propriété Text d'un objet **Label**.

```
warningLabel.Text = "Data not saved"
```

Champs et propriétés

Les *champs* et les *propriétés* sont des informations stockées dans un objet. Vous récupérez et définissez leurs valeurs avec des instructions d'assignation de la même façon que vous récupérez et définissez des variables locales dans une procédure. L'exemple suivant récupère la propriété Width et définit la propriété ForeColor d'un objet **Label**.

```
Dim warningWidth As Integer = warningLabel.Width
warningLabel.ForeColor = System.Drawing.Color.Red
```

Notez qu'un champ porte également le nom de *variable membre*.

Pour plus d'informations, consultez Procédures Property et champs.

Méthodes

Une *méthode* est une action que peut effectuer un objet. Par exemple, Add est une méthode de l'objet ComboBox qui ajoute une nouvelle entrée à une zone de liste.

L'exemple suivant présente une utilisation de la méthode Start d'un objet Timer.

```
Dim safetyTimer As New System.Windows.Forms.Timer
safetyTimer.Start()
```

Notez qu'une méthode est simplement une *procédure* qui est exposée par un objet.

Pour plus d'informations, consultez Comment : exécuter des actions avec les méthodes.

Événements

Un événement est une action reconnue par un objet, telle qu'un clic de souris ou une pression sur une touche, pour laquelle vous pouvez écrire du code pour y répondre. Les événements peuvent se produire suite à une action d'un utilisateur ou au code de programme, ou être provoqués par le système. Le code qui signale un événement *déclenche* l'événement et le code qui y répond *gère* l'événement.

Vous pouvez également développer vos événements personnalisés déclenchés par vos objets et gérés par d'autres objets. Pour plus d'informations, consultez Événements et gestionnaires d'événements.

Membres d'instance et membres partagés

Lorsque vous créez un objet à partir d'une classe, le résultat est une instance de cette classe. Les membres qui ne sont pas déclarés avec le mot clé Shared (Visual Basic) sont des *membres d'instance*, qui appartiennent strictement à cette instance particulière. Un membre d'instance dans une instance est indépendant du même membre dans une autre instance de la même classe. Par exemple, une variable membre d'instance peut avoir des valeurs différentes dans différentes instances.

Les membres déclarés avec le mot clé **Shared** sont des *membres partagés*, qui appartiennent à la classe dans son ensemble et non à une instance particulière. Un membre partagé existe uniquement une fois, quel que soit le nombre d'instances de sa classe que vous créez, ou même si vous ne créez pas d'instance. Par exemple, une variable membre partagée possède une seule valeur, qui est disponible à tout le code qui peut accéder à la classe.

Liste de membres IntelliSense

IntelliSense répertorie les membres d'une classe lorsque vous appelez son option Liste des membres, par exemple lorsque vous tapez un point (**.**) comme opérateur d'accès de membre. Si vous tapez le point après le nom d'une variable déclarée comme une instance de cette classe, IntelliSense répertorie tous les membres d'instance et aucun des membres partagés. Si vous tapez le point après le nom de classe lui-même, IntelliSense répertorie tous les membres partagés et aucun des membres d'instance. Pour plus d'informations, consultez Utilisation de la fonctionnalité IntelliSense.

Voir aussi

Concepts

Relations entre les objets

Autres ressources

Objets en Visual Basic

Comment : accéder aux membres partagés et non partagés d'un objet

Une fois qu'un objet est créé, vous pouvez accéder à ses champs, propriétés, méthodes et événements par le biais de la variable de l'objet. Si le membre est Shared (Visual Basic), vous n'êtes pas obligé de créer un objet pour y accéder.

Accès à des membres non partagés

Pour accéder à un membre non partagé d'un objet

1. Assurez-vous que l'objet a été créé à partir de sa classe et assigné à une variable objet.

```
Dim secondForm As New System.Windows.Forms.Form
```

2. Dans l'instruction qui accède au membre, faites suivre le nom de la variable objet de l'*opérateur d'accès de membre* (**.**), puis du nom du membre.

```
secondForm.Show()
```

Accès à des membres partagés

Pour accéder à un membre partagé d'un objet

- Faites suivre le nom de classe de l'*opérateur d'accès de membre* (**.**), puis du nom du membre. Vous devez toujours accéder à un membre **Shared** de l'objet directement par le biais du nom de classe.

```
MsgBox("This computer is called " & Environment.MachineName)
```

- Si vous avez déjà créé un objet à partir de la classe, vous pouvez également accéder à un membre **Shared** par le biais de la variable de l'objet.

Voir aussi

Tâches

Comment : créer un objet
Comment : réutiliser un composant existant
Comment : définir une classe qui utilise les membres d'une classe existante
Comment : accéder aux membres d'un objet

Concepts

Membres objets

Chaînes en Visual Basic

Cette section décrit les concepts de base de l'utilisation de chaînes en Visual Basic.

Dans cette section

Introduction aux chaînes dans Visual Basic

Répertorie les rubriques qui décrivent les concepts de base de l'utilisation de chaînes en Visual Basic.

Génération de chaînes en Visual Basic

Répertorie les rubriques qui expliquent comment créer de nouvelles chaînes.

Conversion entre chaînes et d'autres types de données en Visual Basic

Répertorie les rubriques qui expliquent comment convertir des chaînes en d'autres types de données.

Procédures pas à pas pour les chaînes en Visual Basic

Répertorie les procédures pas à pas impliquant la manipulation de chaînes.

Validation de chaînes dans Visual Basic

Répertorie les rubriques qui expliquent comment valider des chaînes.

Recherche et remplacement de chaînes en Visual Basic

Répertorie les rubriques qui expliquent comment utiliser les fonctions de recherche et de remplacement pour les chaînes.

Analyse de chaînes en Visual Basic

Répertorie les rubriques qui expliquent comment analyser des chaînes.

Dépannage des chaînes en Visual Basic

Répertorie les rubriques qui fournissent des conseils sur le dépannage de la manipulation de chaînes.

Voir aussi

Autres ressources

Fonctionnalités du langage Visual Basic

Introduction aux chaînes dans Visual Basic

Cette section décrit les concepts de base qui sous-tendent l'utilisation des chaînes en Visual Basic.

Dans cette section

Concepts de base des chaînes en Visual Basic

Présente les concepts de base qui sous-tendent l'utilisation des chaînes et des variables chaîne.

Types de méthodes de manipulation de chaînes en Visual Basic

Présente de nombreuses méthodes permettant d'analyser et de manipuler les chaînes.

Nothing et les chaînes en Visual Basic

Décrit comment le runtime Visual Basic et le .NET Framework évaluent **Nothing** différemment lorsqu'il s'agit de chaînes.

Comment les informations de culture affectent les chaînes dans Visual Basic

Décrit comment Visual Basic utilise les informations de culture pour effectuer des conversions de chaînes et des comparaisons.

Accès à une chaîne de base zéro et accès à une chaîne de base un en Visual Basic

Compare la façon dont Visual Basic et le .NET Framework permettent d'accéder aux caractères d'une chaîne.

Voir aussi

Autres ressources

Chaînes en Visual Basic
Génération de chaînes en Visual Basic

Concepts de base des chaînes en Visual Basic

Le type de données **String** représente un ensemble de caractères (chacun représentant à son tour une instance du type de données **Char**). Cette rubrique présente les concepts de base des chaînes en Visual Basic.

Variables chaîne

Une instance d'une chaîne (String) peut être assignée à une valeur littérale qui représente un ensemble de caractères. Par exemple :

VB

```
Dim MyString As String
MyString = "This is an example of the String data type"
```

Une variable **String** peut également accepter une expression qui a pour valeur une chaîne. Voici quelques exemples :

VB

```
Dim OneString As String
Dim TwoString As String
OneString = "one, two, three, four, five"
' Evaluates to "two".
TwoString = OneString.Substring(5, 3)
OneString = "1"
' Evaluates to "11".
TwoString = OneString & "1"
```

Tout littéral assigné à une variable **String** doit être inséré entre guillemets (""). Cela signifie qu'un guillemet dans une chaîne ne peut pas être représenté par un guillemet. Par exemple, le code suivant crée une erreur de compilation :

VB

```
Dim myString As String

' This line would cause an error.
' myString = "He said, "Look at this example!""
```

Ce code crée une erreur car le compilateur arrête la chaîne après le deuxième guillemet, et le reste de la chaîne est interprété comme du code. Pour résoudre ce problème, Visual Basic interprète deux guillemets dans un littéral de chaîne comme un guillemet dans la chaîne. L'exemple suivant illustre la façon correcte de placer une chaîne entre guillemets :

VB

```
' The value of myString is: He said, "Look at this example!"
myString = "He said, ""Look at this example!"""
```

Dans l'exemple précédent, les deux guillemets qui précèdent le mot `Look` deviennent un guillemet dans la chaîne. Les trois guillemets à la fin de la ligne représentent un guillemet de la chaîne et le caractère de fin de la chaîne.

Caractères des chaînes

Une chaîne peut être perçue comme un ensemble de valeurs **Char**. Par ailleurs, le type **String** comprend des fonctions intégrées qui permettent d'effectuer de nombreuses manipulations sur une chaîne, semblables aux manipulations autorisées par les tableaux. Comme tous les tableaux du .NET Framework, ce sont des tableaux de base zéro. Vous pouvez faire référence à un caractère spécifique d'une chaîne par l'intermédiaire de la propriété **Chars**, qui permet d'accéder à un caractère grâce à sa position dans la chaîne. Par exemple :

VB

```
Dim myString As String = "ABCDE"
```

```
Dim myChar As Char
' The value of myChar is "C".
myChar = myString.Chars(3)
```

Dans l'exemple précédent, la propriété **Chars** de la chaîne retourne le quatrième caractère de la chaîne, `D`, et l'assigne à `myChar`. La propriété **Length** vous permet d'obtenir la longueur d'une chaîne particulière. Si vous devez exécuter plusieurs manipulations de type tableau sur une chaîne, vous pouvez la convertir en un tableau d'instances de **Char** à l'aide de la fonction **ToCharArray** de la chaîne. Par exemple :

VB

```
Dim myString As String = "abcdefghijklmnop"
Dim myArray As Char() = myString.ToCharArray
```

La variable `myArray` contient à présent un tableau composé de valeurs **Char**, chacune représentant un caractère issu de `myString`.

Immuabilité des chaînes

Une chaîne est *immuable*, c'est-à-dire que sa valeur ne peut pas être modifiée après sa création. Toutefois, cela ne vous empêche pas d'assigner plusieurs valeurs à une variable chaîne. Prenons l'exemple suivant :

VB

```
Dim myString As String = "This string is immutable"
myString = "Or is it?"
```

Dans cet exemple, une variable chaîne est créée, une valeur lui est affectée, puis sa valeur est modifiée.

Plus spécifiquement, une instance de type **String** est créée dans la première ligne, et la valeur `This string is immutable` lui est affectée. Dans la deuxième ligne de l'exemple, une nouvelle instance est créée et la valeur `Or is it?` lui est affectée. La variable chaîne ignore sa référence à la première instance et stocke une référence à la nouvelle instance.

Contrairement à d'autres types de données intrinsèques, **String** est un type référence. Lorsqu'une variable de type référence est passée en tant qu'argument à une fonction ou à une sous-routine, une référence à l'adresse mémoire où sont stockées les données est passée à la place de la valeur réelle de la chaîne. Dans l'exemple précédent, le nom de la variable reste donc identique, mais pointe vers une nouvelle instance différente de la classe **String**, qui contient la nouvelle valeur.

Voir aussi

Référence

String, type de données (Visual Basic)

Char, type de données (Visual Basic)

Autres ressources

Introduction aux chaînes dans Visual Basic

Opérations de chaînes de base

Types de méthodes de manipulation de chaînes en Visual Basic

Il existe plusieurs méthodes différentes pour analyser et manipuler vos chaînes. Certaines de ces méthodes font partie du langage Visual Basic, tandis que d'autres sont inhérentes à la classe **String**.

Langage Visual Basic et le .NET Framework

Les méthodes Visual Basic sont utilisées en tant que fonctions inhérentes du langage. Elles peuvent être utilisées sans qualification dans votre code. L'exemple suivant illustre une utilisation classique d'une commande de manipulation de chaîne Visual Basic :

VB

```
Dim aString As String = "SomeString"
Dim bString As String
bString = Mid(aString, 3, 3)
```

Dans cet exemple, la fonction **Mid** effectue une opération directe sur `aString` et assigne la valeur à `bString`.

Méthodes partagées et méthodes d'instance

Vous pouvez également manipuler des chaînes avec les méthodes de la classe **String**. Les deux types de méthodes de **String** sont les suivantes : les méthodes *partagées* et les méthodes d'*instance*.

Méthodes partagées

Une méthode partagée est issue de la classe **String** et ne nécessite pas une instance de cette classe. Ces méthodes peuvent être désignées par le nom de la classe (**String**) plutôt que par une instance de la classe **String**. Par exemple :

VB

```
Dim aString As String = String.Copy("A literal string")
```

Dans l'exemple précédent, la méthode System.String.Copy(System.String) est une méthode statique, qui agit sur une expression qui lui est attribuée et qui assigne le résultat à `bString`.

Méthodes d'instance

En revanche, les méthodes d'instance proviennent d'une instance particulière de **String** et doivent être désignées par le nom de l'instance. Par exemple :

VB

```
Dim aString As String = "A String"
Dim bString As String
' Assign "String" to bString.
bString = aString.Substring(2, 6)
```

Dans cet exemple, la méthode System.String.Substring(System.Int32) est une méthode de l'instance de **String** (c'est-à-dire `aString`). Elle effectue une opération sur `aString` et assigne la valeur à `bString`.

Voir aussi

Autres ressources

Introduction aux chaînes dans Visual Basic

Nothing et les chaînes en Visual Basic

Le runtime Visual Basic et le .NET Framework évaluent **Nothing** différemment lorsqu'il s'agit de chaînes.

Runtime Visual Basic et le .NET Framework

Prenons l'exemple suivant :

VB

```
Dim MyString As String = "This is my string"
Dim stringLength As Integer
' Explicitly set the string to Nothing.
MyString = Nothing
' stringLength = 0
stringLength = Len(MyString)
' This line, however, causes an exception to be thrown.
stringLength = MyString.Length
```

Le runtime Visual Basic évalue habituellement **Nothing** comme une chaîne vide (""). Ce n'est pas le cas du .NET Framework qui lève une exception lors d'une tentative d'exécution d'une opération de chaîne sur **Nothing**.

Voir aussi

Autres ressources

Introduction aux chaînes dans Visual Basic

Comment les informations de culture affectent les chaînes dans Visual Basic

Cette page d'aide décrit comment Visual Basic utilise les informations de culture pour effectuer des conversions et des comparaisons de chaînes.

Période à laquelle utiliser les chaînes spécifiques à la culture

En général, vous devez utiliser des chaînes spécifiques à la culture pour toutes les données présentées aux utilisateurs et lues par ces derniers, et utiliser des chaînes de culture dites indifférentes pour les données internes de votre application.

Par exemple, si votre application demande aux utilisateurs d'entrer une date en tant que chaîne, elle doit s'attendre à ce que les utilisateurs mettent en forme les chaînes en fonction de leur culture et l'application doit convertir la chaîne en conséquence. Si votre application présente ensuite la date dans son interface utilisateur, elle doit le faire dans la culture de l'utilisateur.

Toutefois, si l'application télécharge la date vers un serveur central, elle doit mettre en forme la chaîne en fonction d'une culture spécifique pour éviter la confusion entre des formats de date potentiellement différents.

Fonctions dépendantes de la culture

Toutes les fonctions de conversion de chaînes de Visual Basic (à l'exception des fonctions **Str** et **Val**) utilisent les informations de culture de l'application pour vérifier que les conversions et les comparaisons sont conformes à la culture de l'utilisateur de l'application.

Pour utiliser avec succès les fonctions de conversion de chaînes dans les applications qui s'exécutent sur les ordinateurs ayant des paramètres de culture différents, vous devez connaître les fonctions qui utilisent un paramètre de culture spécifique et celles qui utilisent le paramètre de culture actuel. Notez que les paramètres de culture de l'application sont, par défaut, hérités des paramètres de culture du système d'exploitation. Pour plus d'informations, consultez Fonctions Asc, AscW, Fonctions Chr, ChrW, Format, fonction, Hex, fonction (Visual Basic), Oct, fonction et Fonctions de conversion de types de données.

Les fonctions **Str** (convertit des nombres en chaînes) et **Val** (convertit des chaînes en nombres) n'utilisent pas les informations de culture de l'application lors de la conversion entre les chaînes et les nombres. Elles ne reconnaissent en fait que le point (.) comme séparateur décimal valide. Les analogues de ces fonctions sensibles à l'environnement culturel sont les suivants :

- **conversions qui utilisent la culture actuelle.** Les fonctions **CStr** et **Format** convertissent un nombre en une chaîne et les fonctions **CDbl** et **CInt** convertissent une chaîne en un nombre ;
- **conversions qui utilisent une culture spécifique.** Chaque objet numérique a une méthode `ToString(IFormatProvider)` qui convertit un nombre en une chaîne, et une méthode `Parse(String, IFormatProvider)` qui convertit une chaîne en un nombre. Par exemple, le type **Double** fournit les méthodes ToString et Parse.

Pour plus d'informations, consultez Str, fonction et Val, fonction.

Utilisation d'une culture spécifique

Supposons que vous développez une application qui envoie une date (mise en forme comme une chaîne) à un service Web. Dans ce cas, votre application doit utiliser une culture spécifique pour la conversion de la chaîne. Pour en connaître la raison, prenez le résultat de l'utilisation de la méthode ToString de la date : si votre application utilise cette méthode pour mettre en forme la date du 4 juillet 2005, elle retourne « 7/4/2005 12:00:00 AM »" lorsqu'elle est exécutée avec la culture Anglo-américain (en-USA), mais elle retourne « 04.07.2005 00:00:00 » lorsqu'elle est exécutée avec la culture Allemand (de-DE).

Lorsque vous devez exécuter la conversion d'une chaîne en un format de culture spécifique, vous devez utiliser la classe **CultureInfo** qui est intégrée dans le .NET Framework. Vous pouvez créer un objet **CultureInfo** pour une culture spécifique en passant le nom de la culture au constructeur CultureInfo. Les noms de cultures pris en charge sont répertoriés dans la page d'aide de la classe CultureInfo.

Vous pouvez également obtenir une instance de la *culture dite indifférente* à partir de la propriété System.Globalization.CultureInfo.InvariantCulture. La culture dite indifférente est basée sur la culture anglaise, mais il existe certaines différences. Par exemple, elle spécifie une horloge de 24 heures au lieu d'une horloge de 12 heures.

Pour remplacer une date par la chaîne de culture, passez l'objet **CultureInfo** à la méthode ToString de l'objet de la date. Par exemple, le code suivant affiche « 07/04/2005 00:00:00 », quels que soient les paramètres de culture de l'application.

```
Dim d As Date = #7/4/2005#
MsgBox(d.ToString(System.Globalization.CultureInfo.InvariantCulture))
```

Remarque

Les littéraux de date sont toujours interprétés en fonction de la culture anglophone.

Comparaison de chaînes

Il existe deux situations importantes dans lesquelles les comparaisons de chaînes sont nécessaires :

- **triage des données à afficher pour l'utilisateur.** Utilisez les opérations en fonction de la culture actuelle afin que les chaînes puissent être triées correctement ;
- **détermination de la correspondance exacte ou non entre deux chaînes internes d'une application (généralement pour des raisons de sécurité).** Utilisez des opérations qui ignorent la culture actuelle.

Vous pouvez exécuter les deux types de comparaisons avec la fonction Visual Basic StrComp. Spécifiez l'argument facultatif *Compare* pour contrôler le type de comparaison : Text pour la plupart des entrées et sorties, Binary pour déterminer les correspondances exactes.

La fonction **StrComp** retourne un entier qui indique la relation entre les deux chaînes comparées selon l'ordre de tri. Une valeur positive du résultat indique que la première chaîne est plus grande que la seconde. Un résultat négatif indique que la première chaîne est plus petite, tandis que la valeur zéro indique une égalité entre les chaînes.

VB

```
' Defines variables.
Dim TestStr1 As String = "ABCD"
Dim TestStr2 As String = "abcd"
Dim TestComp As Integer
' The two strings sort equally. Returns 0.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Text)
' TestStr1 sorts after TestStr2. Returns -1.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Binary)
' TestStr2 sorts before TestStr1. Returns 1.
TestComp = StrComp(TestStr2, TestStr1)
```

Vous pouvez également utiliser le partenaire du .NET Framework de la fonction **StrComp**, la méthode System.String.Compare(System.String,System.String). C'est une méthode statique et surchargée de la classe String de base. L'exemple suivant illustre l'utilisation de cette méthode :

VB

```
Dim myString As String = "Alphabetical"
Dim secondString As String = "Order"
Dim result As Integer
result = String.Compare(myString, secondString)
```

Pour un contrôle plus fin de la manière dont les comparaisons sont effectuées, vous pouvez utiliser des surcharges supplémentaires de la méthode **Compare**. Avec la méthode System.String.Compare(System.String,System.String,System.StringComparison), vous pouvez utiliser l'argument *comparisonType* pour spécifier le type de comparaison à utiliser.

Valeur de l'argument *comparisonType*	**Type de comparaison**	**Quand l'utiliser**
Ordinal	Comparaison en fonction des octets de composant des chaînes.	Utilisez cette valeur lors de la comparaison des identificateurs respectant les majuscules et les minuscules, des paramètres relatifs à la sécurité ou des autres identificateurs non linguistiques où les octets doivent correspondre exactement.

OrdinalIgnoreCase	Comparaison en fonction des octets de composant des chaînes. **OrdinalIgnoreCase** utilise les informations de culture dite indifférente pour déterminer à quel moment deux caractères ne diffèrent que dans la mise en majuscules.	Utilisez cette valeur lors de la comparaison des identificateurs ne respectant pas les majuscules et les minuscules, des paramètres relatifs à la sécurité et des données stockées dans Windows.
CurrentCulture ou CurrentCultureIgnoreCase	Comparaison en fonction de l'interprétation des chaînes dans la culture actuelle.	Utilisez ces valeurs lors de la comparaison des données affichées pour l'utilisateur, de la plupart des entrées d'utilisateur et des autres données qui nécessitent une interprétation linguistique.
InvariantCulture ou InvariantCultureIgnoreCase	Comparaison en fonction de l'interprétation des chaînes dans la culture dite indifférente. Elle est différente de **Ordinal** et de **OrdinalIgnoreCase** parce que la culture dite indifférente traite les caractères en dehors de sa plage admise comme des caractères indifférents équivalents.	Utilisez ces valeurs uniquement lors de la comparaison de la persistance des données ou de l'affichage des données linguistiquement appropriées nécessitant un ordre de tri fixe.

Considérations à propos de la sécurité

Si votre application prend des décisions relatives à la sécurité en fonction du résultat d'une comparaison ou d'une opération de modification des majuscules et des minuscules, l'opération doit utiliser la méthode **System.String.Compare(System.String,System.String,System.StringComparison)** et passer **Ordinal** ou **OrdinalIgnoreCase** pour l'argument *comparisonType*.

Voir aussi

Référence

Fonctions de conversion de types de données

CultureInfo

Autres ressources

Introduction aux chaînes dans Visual Basic

Accès à une chaîne de base zéro et accès à une chaîne de base un en Visual Basic

Cette rubrique compare comment Visual Basic et le .NET Framework fournissent un accès aux caractères d'une chaîne. Le .NET Framework fournit toujours un accès de base zéro aux caractères d'une chaîne, tandis que Visual Basic fournit un accès de base zéro et de base 1, selon la fonction.

Base 1

Pour obtenir un exemple d'une fonction Visual Basic de base 1, prenez la fonction **Mid**. Elle accepte un argument qui indique la position de caractère où la sous-chaîne démarrera, en commençant à la position 1. La méthode System.String.Substring(System.Int32) du .NET Framework accepte un index du caractère de la chaîne où la sous-chaîne doit démarrer, en commençant à la position 0. Ainsi, si vous disposez d'une chaîne « ABCDE », les caractères sont numérotés 1,2,3,4,5 pour une utilisation avec la fonction **Mid**, mais 0,1,2,3,4 pour une utilisation avec la méthode **System.String.Substring(System.Int32)**.

Base zéro

Pour obtenir un exemple d'une fonction Visual Basic de base zéro, prenez la fonction **Split**. Elle fractionne une chaîne et retourne un tableau contenant les sous-chaînes. La méthode System.String.Split(System.Char[]) du .NET Framework fractionne également une chaîne et retourne un tableau contenant les sous-chaînes. Étant donné que la fonction **Split** et la méthode **Split** retournent les tableaux du .NET Framework, elles doivent être de base zéro.

Voir aussi

Tâches

Dépannage des collections

Référence

Mid, fonction (Visual Basic)
Split, fonction (Visual Basic)
Substring
Split

Autres ressources

Introduction aux chaînes dans Visual Basic

Génération de chaînes en Visual Basic

Cette section explique comment construire des chaînes en Visual Basic.

Dans cette section

Comment : créer des chaînes à l'aide de StringBuilder en Visual Basic

Construit une longue chaîne à partir de plusieurs chaînes plus petites à l'aide de la classe StringBuilder.

Comment : générer des littéraux de chaîne multilignes (Visual Basic)

Construit un littéral de chaîne d'une longueur de trois lignes.

Voir aussi

Concepts

Utilisation de la classe StringBuilder

Comment : créer des chaînes à l'aide de StringBuilder en Visual Basic

Cet exemple construit une longue chaîne à partir de plusieurs chaînes plus petites à l'aide de la classe StringBuilder. La classe **StringBuilder** est plus efficace que l'opérateur **&=** pour concaténer plusieurs chaînes.

Exemple

L'exemple suivant crée une instance de la classe **StringBuilder**, ajoute 1 000 chaînes à cette instance, puis retourne sa représentation sous forme de chaîne.

VB

```
Private Function StringBuilderTest() As String
    Dim builder As New System.Text.StringBuilder
    For i As Integer = 1 To 1000
        builder.Append("Step " & i & vbCrLf)
    Next
    Return builder.ToString
End Function
```

Voir aussi

Référence

&=, opérateur (Visual Basic)

Concepts

Utilisation de la classe StringBuilder
Création de nouvelles chaînes
Strings, exemple

Autres ressources

Chaînes en Visual Basic
Manipulation des chaînes
Génération de chaînes en Visual Basic

Comment : générer des littéraux de chaîne multilignes (Visual Basic)

Cet exemple construit un littéral de chaîne d'une longueur de trois lignes.

Exemple

VB

```
Dim MyString As String
MyString = "This is the first line of my string." & VbCrLf & _
           "This is the second line of my string." & VbCrLf & _
           "This is the third line of my string."
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve dans **Types de données - définis par Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite :

- Une instruction **Imports** spécifiant l'espace de noms System. Pour plus d'informations, consultez Imports, instruction.

Voir aussi

Concepts

Conversion de chaînes en types de données .NET Framework
Création de nouvelles chaînes
Strings, exemple

Autres ressources

Chaînes en Visual Basic
Analyse de chaînes
Manipulation des chaînes
Génération de chaînes en Visual Basic

Conversion entre chaînes et d'autres types de données en Visual Basic

Cette section décrit comment convertir des chaînes en d'autres types de données.

Dans cette section

Comment : convertir un tableau d'octets en chaîne en Visual Basic

Comment convertir les octets d'un tableau d'octets en une chaîne.

Comment : convertir des chaînes en tableau d'octets en Visual Basic

Comment convertir une chaîne en un tableau d'octets.

Comment : créer une chaîne à partir d'un tableau de valeurs Char (Visual Basic)

Comment créer la chaîne "abcd" en utilisant différents caractères.

Comment : convertir des chaînes hexadécimales en nombres

Comment convertir une chaîne hexadécimale en un entier.

Comment : convertir un tableau d'octets en chaîne en Visual Basic

Cette rubrique montre comment convertir les octets d'un tableau d'octets en une chaîne.

Exemple

Cet exemple utilise la méthode GetString de la classe de codage System.Text.Encoding.Unicode pour convertir tous les octets d'un tableau d'octets en une chaîne.

VB

```
Private Function UnicodeBytesToString( _
    ByVal bytes() As Byte) _
    As String

    Return System.Text.Encoding.Unicode.GetString(bytes)
End Function
```

Vous pouvez choisir entre plusieurs options de codage pour convertir un tableau d'octets en une chaîne :

- System.Text.Encoding.ASCII : obtient un codage pour le jeu de caractères ASCII (7 bits).
- System.Text.Encoding.BigEndianUnicode : obtient un codage pour le format UTF-16 qui utilise l'ordre d'octet avec primauté des octets de poids fort (Big-endian).
- System.Text.Encoding.Default : obtient un codage pour la page de codes ANSI actuelle du système.
- **System.Text.Encoding.Unicode** : obtient un codage pour le format UTF-16 qui utilise l'ordre d'octet avec primauté des octets de poids faible (Big-endian).
- System.Text.Encoding.UTF32 : obtient un codage pour le format UTF-32 qui utilise l'ordre d'octet avec primauté des octets de poids faible (Big-endian).
- System.Text.Encoding.UTF7 : obtient un codage pour le format UTF-7.
- System.Text.Encoding.UTF8 : obtient un codage pour le format UTF-8.

Voir aussi

Tâches

Comment : convertir des chaînes en tableau d'octets en Visual Basic

Référence

System.Text.Encoding

GetString

Comment : convertir des chaînes en tableau d'octets en Visual Basic

Cette rubrique montre comment convertir une chaîne en un tableau d'octets.

Exemple

Cet exemple utilise la méthode GetBytes de la classe de codage System.Text.Encoding.Unicode pour convertir une chaîne en un tableau d'octets.

VB

```
Private Function UnicodeStringToBytes( _
    ByVal str As String) _
    As Byte()

    Return System.Text.Encoding.Unicode.GetBytes(str)
End Function
```

Vous pouvez choisir entre plusieurs options de codage pour convertir une chaîne en un tableau d'octets :

- System.Text.Encoding.ASCII : obtient un codage pour le jeu de caractères ASCII (7 bits).
- System.Text.Encoding.BigEndianUnicode : obtient un codage pour le format UTF-16 qui utilise l'ordre d'octet avec primauté des octets de poids fort (Big-endian).
- System.Text.Encoding.Default : obtient un codage pour la page de codes ANSI actuelle du système.
- **System.Text.Encoding.Unicode** : obtient un codage pour le format UTF-16 qui utilise l'ordre d'octet avec primauté des octets de poids faible (Big-endian).
- System.Text.Encoding.UTF32 : obtient un codage pour le format UTF-32 qui utilise l'ordre d'octet avec primauté des octets de poids faible (Big-endian).
- System.Text.Encoding.UTF7 : obtient un codage pour le format UTF-7.
- System.Text.Encoding.UTF8 : obtient un codage pour le format UTF-8.

Voir aussi

Tâches

Comment : convertir un tableau d'octets en chaîne en Visual Basic

Référence

System.Text.Encoding

GetBytes

Comment : créer une chaîne à partir d'un tableau de valeurs Char (Visual Basic)

Cet exemple crée la chaîne "abcd" en utilisant des caractères isolés.

Exemple

VB

```
Private Sub MakeStringFromCharacters()
    Dim characters() As Char = {"a"c, "b"c, "c"c, "d"c}
    Dim alphabet As New String(characters)
End Sub
```

Compilation du code

Cette méthode n'a aucune exigence particulière.

La syntaxe `"a"c`, où un seul `c` suit un caractère unique entre guillemets, est utilisé pour créer un littéral de caractère.

Programmation fiable

Les caractères Null (`Chr(0)`) faisant partie de la chaîne produisent des résultats inattendus lors de l'utilisation de la chaîne. Le caractère Null sera inclus dans la chaîne, mais dans certains cas les caractères suivants ne seront pas affichés.

Voir aussi

Référence

Char, type de données (Visual Basic)

String

Concepts

Types de données en Visual Basic

Comment : convertir des chaînes hexadécimales en nombres

Cet exemple convertit une chaîne hexadécimale en un entier à l'aide de la méthode ToInt32.

Pour convertir une chaîne hexadécimale en un nombre

- Utilisez la méthode **ToInt32** pour convertir le nombre exprimé en base-16 en un entier.

  Le premier argument de la méthode **ToInt32** est la chaîne à convertir. Le deuxième argument décrit la base dans laquelle le nombre est exprimé ; hexadécimal est de base 16.

  VB

```
' Assign the value 49153 to i.
Dim i As Integer = Convert.ToInt32("c001", 16)
```

Voir aussi

Référence

Hex, fonction (Visual Basic)
ToInt32

Comment : convertir une chaîne en tableau de caractères en Visual Basic

Il est parfois utile de posséder des données sur les caractères de votre chaîne ainsi que sur les positions de ces caractères dans la chaîne, lorsque vous analysez une chaîne, par exemple. Cet exemple montre comment vous pouvez obtenir un tableau des caractères d'une chaîne en appelant la méthode ToCharArray de la chaîne.

Exemple

Cet exemple montre comment fractionner une chaîne en tableau **Char**, et comment fractionner une chaîne en tableau **String** de ses caractères de texte Unicode. Cette distinction s'explique par le fait que les caractères de texte Unicode peuvent être composés d'au moins deux caractères **Char** (par exemple, une paire de substitution ou une séquence de caractères d'association). Pour plus d'informations, consultez TextElementEnumerator et « The Unicode Standard » à l'adresse http://www.unicode.org.

VB

```
Dim testString1 As String = "ABC"
' Create an array containing "A", "B", and "C".
Dim charArray() As Char = testString1.ToCharArray
```

Il est plus difficile de fractionner une chaîne en ses caractères de texte Unicode, mais cette procédure est nécessaire si vous avez besoin d'informations sur la représentation visuelle d'une chaîne. Cet exemple utilise la méthode SubstringByTextElements pour obtenir des informations sur les caractères de texte Unicode qui composent une chaîne.

VB

```
' This string is made up of a surrogate pair (high surrogate
' U+D800 and low surrogate U+DC00) and a combining character
' sequence (the letter "a" with the combining grave accent).
Dim testString2 As String = ChrW(&HD800) & ChrW(&HDC00) & "a" & ChrW(&H300)

' Create and initialize a StringInfo object for the string.
Dim si As New System.Globalization.StringInfo(testString2)

' Create and populate the array.
Dim unicodeTestArray(si.LengthInTextElements) As String
For i As Integer = 0 To si.LengthInTextElements - 1
    unicodeTestArray(i) = si.SubstringByTextElements(i, 1)
Next
```

Voir aussi

Tâches

Comment : accéder aux caractères dans les chaînes en Visual Basic

Référence

Chars

System.Globalization.StringInfo

Autres ressources

Conversion entre chaînes et d'autres types de données en Visual Basic

Chaînes en Visual Basic

Comment : accéder aux caractères dans les chaînes en Visual Basic

Cet exemple montre comment utiliser la propriété Chars pour accéder au caractère à l'emplacement spécifié dans une chaîne.

Exemple

Il est parfois utile de posséder des données sur les caractères de votre chaîne ainsi que sur les positions de ces caractères dans la chaîne. Une chaîne peut être perçue comme un tableau de caractères (instances de **Char**) ; vous pouvez récupérer un caractère particulier en faisant référence à l'index de ce caractère par l'intermédiaire de la propriété **Chars**.

VB

```
Dim myString As String = "ABCDE"
Dim myChar As Char
' Assign "D" to myChar.
myChar = myString.Chars(3)
```

Le paramètre *index* de la propriété **Chars** est à base zéro.

Programmation fiable

La propriété **Chars** retourne le caractère à la position spécifiée. Toutefois, certains caractères Unicode peuvent être représentés par plusieurs caractères. Pour plus d'informations sur l'utilisation des caractères Unicode, consultez Comment : convertir une chaîne en tableau de caractères en Visual Basic.

La propriété **Chars** lève une exception IndexOutOfRangeException si le paramètre *index* est supérieur ou égal à la longueur de la chaîne, ou s'il est inférieur à zéro

Voir aussi

Tâches

Comment : convertir une chaîne en tableau de caractères en Visual Basic

Référence

Chars

Autres ressources

Conversion entre chaînes et d'autres types de données en Visual Basic

Chaînes en Visual Basic

Procédures pas à pas pour les chaînes en Visual Basic

Cette section contient une procédure pas à pas relative à la manipulation de chaînes.

Dans cette section

Procédure pas à pas : chiffrement et déchiffrement de chaînes en Visual Basic

Montre comment chiffrer et déchiffrer des chaînes à l'aide de l'algorithme Data Encryption Standard.

Voir aussi

Autres ressources

Chaînes en Visual Basic

Procédure pas à pas : chiffrement et déchiffrement de chaînes en Visual Basic

Cette procédure pas à pas vous explique comment utiliser la classe DESCryptoServiceProvider pour chiffrer et déchiffrer des chaînes à l'aide de la version du fournisseur de services de chiffrement (CSP) de l'algorithme Triple Data Encryption Standard (TripleDES). La première étape consiste à créer une classe wrapper simple qui encapsule l'algorithme 3DES et stocke les données chiffrées sous forme de chaîne codée en base 64. Ensuite, ce wrapper est utilisé pour stocker en toute sécurité les données utilisateur privées dans un fichier texte accessible publiquement.

Vous pouvez utiliser le chiffrement pour protéger des données utilisateur confidentielles (par exemple, des mots de passe) et rendre les informations d'identification illisibles par les utilisateurs non autorisés. Cela peut empêcher le vol de l'identité d'un utilisateur autorisé ce qui protège les ressources de l'utilisateur et garantit la non-répudiabilité. Le chiffrement empêche également l'accès aux données d'un utilisateur par des utilisateurs non autorisés.

Pour plus d'informations, consultez Vue d'ensemble du chiffrement.

Remarque de sécurité

Les algorithmes Rijndael (maintenant appelé Advanced Ecryption Standard [AES]) et Triple Data Encryption Standard (3DES) offrent une plus grande sécurité que DES parce qu'ils sollicitent beaucoup de ressources informatiques. Pour plus d'informations, consultez DES et Rijndael.

Pour créer le wrapper de chiffrement

1. Ajoutez une instruction import de l'espace de noms de chiffrement au début du fichier.

   VB

   ```
   Imports System.Security.Cryptography
   ```

2. Créez une classe pour encapsuler les méthodes de chiffrement et de déchiffrement.

   VB

   ```
   Public NotInheritable Class Simple3Des
   End Class
   ```

3. Ajoutez un champ privé pour stocker le fournisseur de services de chiffrement 3DES.

   VB

   ```
   Private TripleDes As New TripleDESCryptoServiceProvider
   ```

4. Ajoutez une méthode privée qui crée un tableau d'octets d'une longueur spécifiée à partir du hachage de la clé spécifiée.

   VB

   ```
   Private Function TruncateHash( _
       ByVal key As String, _
       ByVal length As Integer) _
       As Byte()

       Dim sha1 As New SHA1CryptoServiceProvider

       ' Hash the key.
       Dim keyBytes() As Byte = _
           System.Text.Encoding.Unicode.GetBytes(key)
   ```

```
    Dim hash() As Byte = sha1.ComputeHash(keyBytes)

    ' Truncate or pad the hash.
    ReDim Preserve hash(length - 1)
    Return hash
End Function
```

5. Ajoutez un constructeur pour initialiser le fournisseur de services de chiffrement 3DES.

   Le paramètre *key* contrôle les méthodes `EncryptData` et `DecryptData`.

   VB

```
Sub New(ByVal key As String)
    ' Initialize the crypto provider.
    TripleDes.Key = TruncateHash(key, TripleDes.KeySize \ 8)
    TripleDes.IV = TruncateHash("", TripleDes.BlockSize \ 8)
End Sub
```

6. Ajoutez une méthode publique qui chiffre une chaîne.

   VB

```
Public Function EncryptData( _
    ByVal plaintext As String) _
    As String

    ' Convert the plaintext string to a byte array.
    Dim plaintextBytes() As Byte = _
        System.Text.Encoding.Unicode.GetBytes(plaintext)

    ' Create the stream.
    Dim ms As New System.IO.MemoryStream
    ' Create the encoder to write to the stream.
    Dim encStream As New CryptoStream(ms, _
        TripleDes.CreateEncryptor(), _
        System.Security.Cryptography.CryptoStreamMode.Write)

    ' Use the crypto stream to write the byte array to the stream.
    encStream.Write(plaintextBytes, 0, plaintextBytes.Length)
    encStream.FlushFinalBlock()

    ' Convert the encrypted stream to a printable string.
    Return Convert.ToBase64String(ms.ToArray)
End Function
```

7. Ajoutez une méthode publique qui déchiffre une chaîne.

   VB

```
Public Function DecryptData( _
    ByVal encryptedtext As String) _
    As String

    ' Convert the encrypted text string to a byte array.
    Dim encryptedBytes() As Byte = Convert.FromBase64String(encryptedtext)

    ' Create the stream.
```

```
    Dim ms As New System.IO.MemoryStream
    ' Create the decoder to write to the stream.
    Dim decStream As New CryptoStream(ms, _
        TripleDes.CreateDecryptor(), _
        System.Security.Cryptography.CryptoStreamMode.Write)

    ' Use the crypto stream to write the byte array to the stream.
    decStream.Write(encryptedBytes, 0, encryptedBytes.Length)
    decStream.FlushFinalBlock()

    ' Convert the plaintext stream to a string.
    Return System.Text.Encoding.Unicode.GetString(ms.ToArray)
End Function
```

La classe wrapper peut maintenant être utilisée pour protéger les ressources de l'utilisateur. Dans cet exemple, elle est utilisée pour stocker en toute sécurité les données utilisateur privées dans un fichier texte accessible publiquement.

Pour tester le wrapper de chiffrement

1. Dans une classe distincte, ajoutez une méthode qui utilise la méthode `EncryptData` du wrapper pour chiffrer une chaîne et l'écrire dans le dossier Mes documents de l'utilisateur.

   VB

   ```
   Sub TestEncoding()
       Dim plainText As String = InputBox("Enter the plain text:")
       Dim password As String = InputBox("Enter the password:")

       Dim wrapper As New Simple3Des(password)
       Dim cipherText As String = wrapper.EncryptData(plainText)

       MsgBox("The cipher text is: " & cipherText)
       My.Computer.FileSystem.WriteAllText( _
           My.Computer.FileSystem.SpecialDirectories.MyDocuments & _
           "\cipherText.txt", cipherText, False)
   End Sub
   ```

2. Ajoutez une méthode qui lit la chaîne chiffrée du dossier Mes documents de l'utilisateur et la déchiffre à l'aide de la méthode `DecryptData` du wrapper.

   VB

   ```
   Sub TestDecoding()
       Dim cipherText As String = My.Computer.FileSystem.ReadAllText( _
           My.Computer.FileSystem.SpecialDirectories.MyDocuments & _
               "\cipherText.txt")
       Dim password As String = InputBox("Enter the password:")
       Dim wrapper As New Simple3Des(password)

       ' DecryptData throws if the wrong password is used.
       Try
           Dim plainText As String = wrapper.DecryptData(cipherText)
           MsgBox("The plain text is: " & plainText)
       Catch ex As System.Security.Cryptography.CryptographicException
           MsgBox("The data could not be decrypted with the password.")
       End Try
   End Sub
   ```

3. Ajoutez le code d'interface utilisateur pour appeler les méthodes `TestEncoding` et `TestDecoding`.
4. Exécutez l'application.

   Lorsque vous testez l'application, notez qu'elle ne déchiffrera pas les données si vous entrez un mot de passe incorrect.

Voir aussi

Référence

System.Security.Cryptography
DESCryptoServiceProvider
DES
TripleDES
Rijndael

Concepts

Vue d'ensemble du chiffrement

Autres ressources

Procédures pas à pas pour les chaînes en Visual Basic

Validation de chaînes dans Visual Basic

Cette section explique comment valider des chaînes en Visual Basic.

Dans cette section

Construction d'une fonction de validation en Visual Basic

Présente les propriétés des fonctions de validation, et explique comment créer et utiliser des fonctions de validation.

Comment : valider des noms de fichiers et des chemins d'accès dans Visual Basic

Explique comment déterminer si une chaîne représente un nom de fichier ou un chemin d'accès.

Comment : valider des chaînes qui représentent des dates ou des heures (Visual Basic)

Explique comment déterminer si une chaîne représente une date valide.

Comment : valider des chaînes qui représentent des adresses de messagerie (Visual Basic)

Explique comment déterminer si une chaîne représente une adresse de messagerie valide.

Comment : valider le texte à afficher dans un navigateur Web en Visual Basic

Explique comment afficher correctement le texte dans un navigateur.

Utilisation d'expressions régulières avec le contrôle MaskedTextBox en Visual Basic

Explique comment convertir des expressions régulières simples pour utiliser le contrôle MaskedTextBox.

Expressions régulières et opérateur Like

Compare l'opérateur **Like** et les expressions régulières.

Procédure pas à pas : validation de la complexité des mots de passe (Visual Basic)

Explique comment déterminer si une chaîne a les caractéristiques d'un mot de passe fort.

Voir aussi

Autres ressources

Chaînes en Visual Basic
Analyse de chaînes en Visual Basic
Recherche et remplacement de chaînes en Visual Basic
MaskedTextBox, contrôle (Windows Forms)

Construction d'une fonction de validation en Visual Basic

Une *fonction de validation* détermine si une chaîne répond ou non à certaines exigences. Cette rubrique présente les propriétés des fonctions de validation, et explique comment créer et utiliser des fonctions de validation.

Types de fonctions de validation

Une chaîne que vous essayez de valider appartient à l'une de trois catégories suivantes :

1. Validité prouvée.
2. Invalidité prouvée.
3. Validité ou invalidité non prouvée.

Il est clair qu'une fonction de validation doit utiliser les chaînes des deux premières catégories ; la troisième catégorie est plus problématique.

Les fonctions de validation peuvent être regroupées en deux catégories, en fonction de la manière dont elles traitent les chaînes de la troisième catégorie :

- **Fonctions de validation conservatrices.** Ces fonctions valident uniquement les chaînes dont la validité peut être prouvée.

  Vous devez utiliser les fonctions de validation conservatrices lorsque votre application dépend de la chaîne pour la sécurité. Par exemple, ne créez pas de filtre qui tente de filtrer uniquement les éléments à refuser ; il est extrêmement difficile de prendre en compte toutes les entrées nuisibles possibles. Créez plutôt un filtre disposant d'une liste définie des entrées possibles.

- **Fonctions de validation permissives.** Ces fonctions valident toutes les chaînes à moins que leur invalidité soit prouvée.

  Dans la plupart des situations qui ne concernent pas la sécurité, par exemple l'enregistrement du profil d'un utilisateur, vous pouvez utiliser les fonctions de validation permissives qui sont plus flexibles et conviviales que les fonctions de validation conservatrices.

Expressions régulières des fonctions de validation

Vous pouvez utiliser la classe d'expression régulière Regex du .NET Framework pour déterminer si une chaîne correspond à un certain modèle ou si elle contient un certain modèle.

Lors de la validation d'une chaîne, l'expression régulière doit commencer par le caractère ^. Il indique au moteur de l'expression régulière de commencer la mise en correspondance du modèle spécifié au début de la chaîne.

Voir aussi

Tâches

Comment : vérifier que des chaînes sont dans un format d'adresse électronique valide

Autres ressources

Validation de chaînes dans Visual Basic

Expressions régulières du .NET Framework

Comment : valider des noms de fichiers et des chemins d'accès dans Visual Basic

Cet exemple retourne une valeur **Boolean** qui indique si une chaîne représente un nom de fichier ou un chemin d'accès. La validation vérifie si le nom contient des caractères qui ne sont pas autorisés par le système de fichiers.

Exemple

VB

```
Function IsValidFileNameOrPath(ByVal name As String) As Boolean
    ' Determines if the name is Nothing.
    If name Is Nothing Then
        Return False
    End If

    ' Determines if there are bad characters in the name.
    For Each badChar As Char In System.IO.Path.GetInvalidPathChars
        If InStr(name, badChar) > 0 Then
            Return False
        End If
    Next

    ' The name passes basic validation.
    Return True
End Function
```

Cet exemple ne vérifie pas si le nom a placé des deux-points incorrectement, ou des répertoires sans nom, ou encore si la longueur du nom dépasse la longueur maximale définie par le système. Il ne vérifie pas non plus si l'application a l'autorisation nécessaire pour accéder à la ressource du système de fichiers avec le nom spécifié.

Voir aussi

Référence

GetInvalidPathChars

Autres ressources

Validation de chaînes dans Visual Basic

Comment : valider des chaînes qui représentent des dates ou des heures (Visual Basic)

L'exemple de code suivant définit une valeur **Boolean** qui indique si une chaîne représente une date ou une heure valide.

Exemple

VB

```
Dim isValidDate As Boolean = IsDate("01/01/03")
Dim isValidTime As Boolean = IsDate("9:30 PM")
```

Compilation du code

Remplacez `("01/01/03")` et `"9:30 PM"` par la date et l'heure que vous souhaitez valider. Vous pouvez remplacer la chaîne par une autre chaîne codée de manière irréversible, à l'aide d'une variable **String** ou d'une méthode qui retourne une chaîne, telle que **InputBox**.

Programmation fiable

Utilisez cette méthode pour valider la chaîne avant d'essayer de convertir **String** en une variable **DateTime**. Vérifiez d'abord la date ou l'heure pour éviter de générer une exception au moment de l'exécution.

Voir aussi

Référence

IsDate, fonction (Visual Basic)

InputBox, fonction (Visual Basic)

Autres ressources

Validation de chaînes dans Visual Basic

Comment : valider des chaînes qui représentent des adresses de messagerie (Visual Basic)

L'exemple de code suivant définit une variable Boolean qui indique si une chaîne représente une adresse de messagerie valide.

Exemple

VB

```
Function ValidateEmail(ByVal email As String) As Boolean
    Dim emailRegex As _
        New System.Text.RegularExpressions.Regex( _
        "^(?<user>[^@]+)@(?<host>.+)$")
    Dim emailMatch As _
        System.Text.RegularExpressions.Match = emailRegex.Match(email)
    Return emailMatch.Success
End Function
```

Compilation du code

Appelez cette méthode en passant la chaîne qui contient une adresse de messagerie.

Programmation fiable

Cette méthode vérifie que les adresses de messagerie sont sous la forme « personne@microsoft.com ».

Utilisez ce code pour valider la chaîne avant d'essayer de l'utiliser comme adresse de messagerie. Cela pourrait empêcher d'autres erreurs au moment de l'exécution.

Voir aussi

Référence

Regex

Comment : valider le texte à afficher dans un navigateur Web en Visual Basic

Cet exemple montre comment utiliser la méthode HtmlEncode pour afficher correctement le texte dans un navigateur. Les caractères d'une chaîne, par exemple <, > et & sont restitués de manière incorrecte à moins qu'ils soient placés dans une séquence d'échappement sous la forme `<`, `>` et `&`.

Exemple

L'exemple de code suivant code une chaîne à afficher dans un navigateur. Il code la chaîne nommée `TestString` qui contient le texte `This is a <Test String>`, et la copie dans la chaîne nommée `EncodedString` en tant que `This is a <Test String>`.

```
Dim TestString As String = "This is a <Test String>"
Dim EncodedString As String = Server.HtmlEncode(TestString)
```

Cet exemple requiert une référence à System.Web.dll.

Voir aussi

Référence

HtmlEncode

Autres ressources

Validation de chaînes dans Visual Basic

Utilisation d'expressions régulières avec le contrôle MaskedTextBox en Visual Basic

Cet exemple montre comment convertir des expressions régulières simples pour utiliser le contrôle MaskedTextBox.

Description du langage de masquage

Le langage de masquage **MaskedTextBox** standard est basé sur celui utilisé par le contrôle **Masked Edit** dans Visual Basic 6.0, et les utilisateurs qui effectuent une migration à partir de cette plate-forme doivent le maîtriser.

La propriété Mask du contrôle **MaskedTextBox** spécifie le masque de saisie à utiliser. Ce masque doit être une chaîne composée d'un ou de plusieurs des éléments de masquage répertoriés dans le tableau suivant.

Élément de masquage	Description	Élément d'expression régulière
0	Tout chiffre entre 0 et 9. Entrée obligatoire.	\d
9	Chiffre ou espace. Entrée facultative.	[\d]?
#	Chiffre ou espace. Entrée facultative. Si cette position reste vide dans le masque, elle est restituée sous la forme d'un espace. Les signes plus (+) et moins (-) sont autorisés.	[\d+-]?
L	Lettre ASCII. Entrée obligatoire.	[a-zA-Z]
?	Lettre ASCII. Entrée facultative.	[a-zA-Z]?
&	Character Entrée obligatoire.	[\p{Ll}\p{Lu}\p{Lt}\p{Lm}\p{Lo}]
C	Character Entrée facultative.	[\p{Ll}\p{Lu}\p{Lt}\p{Lm}\p{Lo}]?
A	Chiffre alphanumérique. Entrée facultative.	\W
.	Espace réservé à la décimale approprié à la culture.	Non disponible.
,	Espace réservé aux milliers approprié à la culture.	Non disponible.
:	Séparateur d'heure approprié à la culture.	Non disponible.
/	Séparateur de date approprié à la culture.	Non disponible.
$	Symbole monétaire approprié à la culture.	Non disponible.
<	Convertit tous les caractères suivants en minuscules.	Non disponible.
>	Convertit tous les caractères suivants en majuscules.	Non disponible.
|	Annule un décalage vers le haut ou vers le bas effectué précédemment.	Non disponible.
\	Remplace un caractère de masque par un littéral. "\\" est la séquence d'échappement pour une barre oblique inverse.	\

Tous les autres caractères.	Littéraux. Tous les éléments qui ne sont pas de masque apparaissent tels quels dans **MaskedTextBox**.	Tous les autres caractères.

Les symboles de la décimale (.), des millièmes (), de l'heure (:), de la date (/) et de la monnaie ($) affichent par défaut les symboles tels que définis par la culture de l'application. Vous pouvez les forcer à afficher les symboles d'une autre culture à l'aide de la propriété FormatProvider.

Expressions régulières et masques

Bien que vous puissiez utiliser des expressions régulières et des masques pour valider les entrées d'utilisateur, ils ne sont pas tout à fait équivalents. Les expressions régulières peuvent exprimer des modèles plus complexes que les masques, mais les masques peuvent exprimer les mêmes informations plus succinctement et dans un format culturellement pertinent.

Le tableau suivant compare quatre expressions régulières et le masque équivalent pour chacune d'elles.

Expression régulière	Masque	Remarques
`\d{2}/\d{2}/\d{4}`	`00/00/0000`	Le caractère `/` dans le masque est un séparateur de date logique, et est présenté à l'utilisateur sous la forme du séparateur de date approprié à la culture actuelle de l'application.
`\d{2}-[A-Z][a-z]{2}-\d{4}`	`00->L<LL-0000`	Date (jour, abréviation du mois et année) au format américain dans lequel l'abréviation à trois lettres du mois est affichée avec une lettre majuscule initiale suivie de deux lettres minuscules.
`(\(\d{3}\)-)?\d{3}-d{4}`	`(999)-000-0000`	Numéro de téléphone aux États-Unis, indicatif régional facultatif. Si l'utilisateur ne souhaite pas entrer les caractères facultatifs, il peut entrer des espaces ou placer le pointeur de la souris directement à la position représentée par le premier 0 dans le masque.
`$\d{6}.00`	`$999,999.00`	Valeur monétaire comprise entre 0 et 999999. Les caractères de la monnaie, des millièmes et de la décimale sont remplacés au moment de l'exécution par leurs équivalents spécifiques à la culture.

Voir aussi

Référence

Mask

MaskedTextBox

Autres ressources

Validation de chaînes dans Visual Basic

MaskedTextBox, contrôle (Windows Forms)

Expressions régulières et opérateur Like

Cette rubrique compare l'opérateur **Like** Visual Basic et les expressions régulières du .NET Framework.

Comparaison de syntaxe

Le tableau suivant compare la syntaxe du langage de spécification de modèle pour l'opérateur **Like** et la syntaxe des expressions régulières.

Syntaxe de l'opérateur Like	Syntaxe d'une expression régulière
Le comportement de l'opérateur **Like** dépend de l'instruction Option Compare. La méthode de comparaison de chaînes par défaut pour chaque fichier source est **Option Compare Binary**.	Les expressions régulières fonctionnent de manière identique, indépendamment de **Option Compare**.
Pour faire correspondre un caractère unique dans *charlist*, utilisez **[***charlist***]**.	Pour faire correspondre un caractère unique dans *charlist*, utilisez **[***charlist***]**.
Pour faire correspondre un caractère unique qui n'est pas contenu dans *charlist*, utilisez **[!***charlist***]**.	Pour faire correspondre un caractère unique qui n'est pas contenu dans *charlist*, utilisez **[^***charlist***]**.
Pour faire correspondre un chiffre unique (0 à 9), utilisez **#**.	Pour faire correspondre un chiffre unique (0 à 9), utilisez la classe de caractères pour les nombres décimaux, **\d**.
Pour faire correspondre un caractère unique, utilisez **?**.	Pour faire correspondre un caractère unique, spécifiez des classes de caractères mutuellement exclusives pour *charlist* dans **[***charlist***]**. Par exemple, **[\s\S]**.
Pour faire correspondre zéro ou plusieurs caractères, utilisez *****	Pour faire correspondre zéro ou plusieurs caractères, spécifiez des classes de caractères mutuellement exclusives pour *charlist* dans **[***charlist***]***. Par exemple, **[\s\S]***.
Pour faire correspondre un caractère spécial *char*, mettez-le entre parenthèses : **[***char***]**.	Pour faire correspondre un caractère spécial *char*, faites-le précéder d'une barre oblique inverse : *****char*.
Pour faire correspondre un caractère dans une plage, utilisez un trait d'union (**–**) pour séparer les limites inférieures et supérieures de la plage dans *charlist*.	Pour faire correspondre un caractère dans une plage, utilisez un trait d'union (**–**) pour séparer les limites inférieures et supérieures de la plage dans *charlist*.

Voir aussi

Référence

Like, opérateur

Autres ressources

Validation de chaînes dans Visual Basic

Expressions régulières du .NET Framework

Procédure pas à pas : validation de la complexité des mots de passe (Visual Basic)

Cette méthode vérifie certaines caractéristiques du mot de passe fort et met à jour un paramètre de chaîne avec les informations sur les vérifications non concluantes.

Les mots de passe peuvent être utilisés dans un système sécurisé pour autoriser un utilisateur. Toutefois, ils doivent être difficiles à deviner par les utilisateurs non autorisés. Des intrus peuvent utiliser un programme d'*attaque de dictionnaire*, qui parcourt tous les mots d'un dictionnaire (ou de plusieurs dictionnaires dans différentes langues) et teste si les mots correspondent au mot de passe d'un utilisateur. Les mots de passe faibles tels que « Yankees » ou « Mustang » peuvent être devinés rapidement. En revanche, les mots de passe plus forts, par exemple « ?You'L1N3vaFiNdMeyeP @ sSWerd! », sont moins faciles à deviner. Un système protégé par mot de passe doit veiller à ce que les utilisateurs choisissent des mots de passe forts.

Un mot de passe fort est complexe (associe des lettres en majuscules et en minuscules, des nombres et des caractères spéciaux) et n'est pas un mot. Cet exemple montre comment vérifier la complexité.

Exemple
VB

```
''' <summary>Determines if a password is sufficiently complex.</summary>
''' <param name="pwd">Password to validate</param>
''' <param name="minLength">Minimum number of password characters.</param>
''' <param name="numUpper">Minimum number of uppercase characters.</param>
''' <param name="numLower">Minimum number of lowercase characters.</param>
''' <param name="numNumbers">Minimum number of numeric characters.</param>
''' <param name="numSpecial">Minimum number of special characters.</param>
''' <returns>True if the password is sufficiently complex.</returns>
Function ValidatePassword(ByVal pwd As String, _
    Optional ByVal minLength As Integer = 8, _
    Optional ByVal numUpper As Integer = 2, _
    Optional ByVal numLower As Integer = 2, _
    Optional ByVal numNumbers As Integer = 2, _
    Optional ByVal numSpecial As Integer = 2) _
    As Boolean

    ' Replace [A-Z] with \p{Lu}, to allow for Unicode uppercase letters.
    Dim upper As New System.Text.RegularExpressions.Regex("[A-Z]")
    Dim lower As New System.Text.RegularExpressions.Regex("[a-z]")
    Dim number As New System.Text.RegularExpressions.Regex("[0-9]")
    ' Special is "none of the above".
    Dim special As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")

    ' Check the length.
    If Len(pwd) < minLength Then Return False
    ' Check for minimum number of occurrences.
    If upper.Matches(pwd).Count < numUpper Then Return False
    If lower.Matches(pwd).Count < numLower Then Return False
    If number.Matches(pwd).Count < numNumbers Then Return False
    If special.Matches(pwd).Count < numSpecial Then Return False

    ' Passed all checks.
    Return True
End Function

Sub TestValidatePassword()
    Dim password As String = "Password"
    ' Demonstrate that "Password" is not complex.
    MsgBox(password & " is complex: " & ValidatePassword(password))

    password = "Z9f%a>2kQ"
    ' Demonstrate that "Z9f%a>2kQ" is not complex.
    MsgBox(password & " is complex: " & ValidatePassword(password))
End Sub
```

Compilation du code

Appelez cette méthode en passant la chaîne qui contient ce mot de passe.

Cet exemple nécessite les éléments suivants :

- l'accès aux membres de l'espace de noms System.Text.RegularExpressions. Ajoutez une instruction **Imports** si vous n'utilisez pas des noms de membres qualifiés complets dans votre code. Pour plus d'informations, consultez Imports, instruction.

Sécurité

Si vous déplacez le mot de passe sur un réseau, vous devez utiliser une méthode sécurisée pour transférer des données. Pour plus d'informations, consultez Sécurité des applications Web ASP.NET.

Vous pouvez améliorer l'exactitude de la fonction `ValidatePassword` en ajoutant des contrôles de complexité supplémentaires :

- Comparez le mot de passe et ses sous-chaînes au nom de l'utilisateur, à l'identificateur de l'utilisateur et à un dictionnaire défini par l'application. En outre, considérez les caractères visuellement semblables comme équivalents lorsque vous effectuez des comparaisons. Par exemple, considérez les lettres « l » et « e » comme équivalentes aux chiffres « 1 » et « 3 ».
- S'il n'existe qu'un seul caractère en majuscule, vérifiez qu'il ne correspond pas au premier caractère du mot de passe.
- Assurez-vous que les deux derniers caractères du mot de passe sont des lettres.
- N'autorisez pas les mots de passe dans lesquels tous les symboles sont saisis à partir de la ligne supérieure du clavier.

Voir aussi

Référence

Regex

Autres ressources

Sécurité des applications Web ASP.NET

Procédures pas à pas pour les chaînes en Visual Basic

Recherche et remplacement de chaînes en Visual Basic

Cette section explique comment utiliser les fonctions de recherche et de remplacement des chaînes en Visual Basic.

Dans cette section

Comment : supprimer des parties d'une chaîne (Visual Basic)

Supprime toutes les occurrences d'une chaîne trouvées dans une autre chaîne.

Comment : rechercher une chaîne dans un tableau de chaînes (Visual Basic)

Rapporte l'index de la première chaîne du tableau avec la sous-chaîne spécifiée et l'index de la sous-chaîne présente dans la chaîne.

Comment : effectuer une recherche dans une chaîne (Visual Basic)

Rapporte l'index de la première occurrence d'une sous-chaîne à l'intérieur d'une chaîne.

Voir aussi

Concepts

Types de méthodes de manipulation de chaînes en Visual Basic

Autres ressources

Chaînes en Visual Basic

Comment : rechercher une chaîne dans un tableau de chaînes (Visual Basic)

Cet exemple effectue une boucle sur chaque chaîne d'un tableau de chaînes pour identifier celles qui contiennent la sous-chaîne spécifiée. Pour chaque correspondance, l'exemple affiche l'index de la sous-chaîne dans la chaîne.

Exemple

L'exemple suivant utilise les méthodes Contains et IndexOf de l'objet **String**.

La méthode **Contains** indique si la chaîne contient la sous-chaîne spécifiée.

La méthode **IndexOf** rapporte l'emplacement du premier caractère de la première occurrence de la sous-chaîne. L'index est basé sur zéro, ce qui signifie que l'index du premier caractère d'une chaîne est 0. Si **IndexOf** ne trouve pas la sous-chaîne, la valeur retournée est -1.

VB

```
Dim StrArray() As String = {"ABCDEFG", "HIJKLMNOP"}
Dim FindThisString As String = "JKL"
For Each Str As String In StrArray
    If Str.Contains(FindThisString) Then
        MsgBox("Found " & FindThisString & " at index " & _
          Str.IndexOf(FindThisString))
    End If
Next
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve dans **Types de données - définis par Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite :

- Une instruction **Imports** spécifiant l'espace de noms System. Pour plus d'informations, consultez Imports, instruction.

Programmation fiable

La méthode **IndexOf** respecte la casse et utilise la culture actuelle.

Pour un contrôle optimal des erreurs, vous pouvez faire figurer la recherche de chaîne dans le bloc **Try** d'une construction Try...Catch...Finally, instruction (Visual Basic).

Voir aussi

Tâches

Comment : effectuer une recherche dans une chaîne (Visual Basic)

Référence

Try...Catch...Finally, instruction (Visual Basic)

IndexOf

Autres ressources

Introduction aux chaînes dans Visual Basic

Comment : effectuer une recherche dans une chaîne (Visual Basic)

Cet exemple appelle la méthode IndexOf sur un objet String pour rapporter l'index de la première occurrence d'une sous-chaîne.

Exemple

VB

```
Dim SearchWithinThis As String = "ABCDEFGHIJKLMNOP"
Dim SearchForThis As String = "DEF"
Dim FirstCharacter As Integer = SearchWithinThis.IndexOf(SearchForThis)
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve dans **Types de données - définis par Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite :

- Une instruction **Imports** spécifiant l'espace de noms System. Pour plus d'informations, consultez Imports, instruction.

Programmation fiable

La méthode **IndexOf** rapporte l'emplacement du premier caractère de la première occurrence de la sous-chaîne. L'index est de base 0, ce qui signifie que l'index du premier caractère d'une chaîne est 0.

Si la méthode **IndexOf** ne trouve pas la sous-chaîne, elle retourne -1.

La méthode **IndexOf** respecte la casse et utilise la culture actuelle.

Pour un contrôle optimal des erreurs, vous pouvez faire figurer la recherche de chaîne dans le bloc **Try** d'une construction Try...Catch...Finally, instruction (Visual Basic).

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

IndexOf

Autres ressources

Introduction aux chaînes dans Visual Basic

Recherche et remplacement de chaînes en Visual Basic

Comment : supprimer des parties d'une chaîne (Visual Basic)

L'exemple de code suivant supprime toutes les occurrences d'une chaîne à partir d'une autre chaîne.

Exemple

Cet exemple utilise la fonction **Replace** pour remplacer chaque occurrence de la sous-chaîne par une chaîne vide. Le résultat est une nouvelle chaîne avec les sous-chaînes supprimées.

Cet exemple assigne la valeur de retour de la fonction **Replace** à la variable chaîne `withoutParts`. La chaîne d'origine, stocké dans `withParts`, n'est pas modifiée.

VB

```
Dim withParts As String = "Books and Chapters and Pages"
Dim withoutParts As String = Replace(withParts, "and ", "")
```

Compilation du code

Remplacez `"Books and Chapters and Pages"` par la chaîne que vous souhaitez manipuler.

Voir aussi

Référence

Replace, fonction (Visual Basic)

String, type de données (Visual Basic)

Analyse de chaînes en Visual Basic

Cette section explique comment analyser des chaînes en Visual Basic.

Dans cette section

Comment : analyser des URI en Visual Basic

Explique comment utiliser la classe System.Uri pour analyser un URI (Uniform Resource Identifier).

Comment : analyser des adresses de messagerie en Visual Basic

Présente une expression régulière simple pour analyser une adresse de messagerie.

Comment : identifier les URL dans une chaîne en Visual Basic

Explique comment utiliser une expression régulière simple pour identifier un URI dans une chaîne.

Comment : identifier le texte d'une chaîne HTML dans Visual Basic

Explique comment utiliser une expression régulière simple pour supprimer des balises d'un document HTML.

Comment : identifier des liens hypertexte dans une chaîne HTML en Visual Basic

Présente une expression régulière simple pour identifier des liens hypertexte dans un document HTML.

Voir aussi

Autres ressources

Chaînes en Visual Basic

Introduction aux chaînes dans Visual Basic

Comment : analyser des URI en Visual Basic

Cet exemple montre comment utiliser la classe System.Uri pour analyser un URI.

Exemple

Cet exemple utilise plusieurs méthodes d'un objet **Uri** pour accéder aux informations sur un URI.

```
Dim uriString As String = "http://www.contoso.com/index.htm?date=today"
Dim uriObject As New Uri(uriString)
' Display "Scheme: http"
MsgBox("Scheme: " & uriObject.Scheme)
' Display "Host: www.contoso.com"
MsgBox("Host: " & uriObject.Host)
' Display "Local path: /index.htm"
MsgBox("Local path: " & uriObject.LocalPath)
' Display "Query: ?date=today"
MsgBox("Query: " & uriObject.Query)
```

Voir aussi

Référence

Uri

Autres ressources

Analyse de chaînes en Visual Basic

Comment : analyser des adresses de messagerie en Visual Basic

Cet exemple présente une expression régulière simple pour analyser une adresse de messagerie.

Exemple

Cet exemple utilise l'expression régulière `(\S+)@([^\.\s]+)(?:\.([^\.\s]+))+`, qui signifie :

1. Ensemble d'un ou plusieurs caractères autres qu'un espace (capturé), suivi de
2. Caractère "@", suivi de
3. Ensemble d'un ou plusieurs caractères autres qu'un espace ou un point (capturé), suivi de
4. Un ou plusieurs des éléments suivants :
    a. Caractère ".", suivi de
    b. Ensemble d'un ou plusieurs caractères autres qu'un espace ou un point (capturé).

La méthode Match de l'expression régulière retourne un objet Match qui contient des informations sur la partie de la chaîne d'entrée qui correspond à l'expression régulière.

```
''' <summary>
''' Parses an e-mail address into its parts.
''' </summary>
''' <param name="emailString">E-mail address to parse.</param>
''' <remarks> For example, this method displays the following
''' text when called with "someone@mail.contoso.com":
''' User name: someone
''' Address part: mail
''' Address part: contoso
''' Address part: com
''' </remarks>
Sub ParseEmailAddress(ByVal emailString As String)
    Dim emailRegEx As New Regex("(\S+)@([^\.\s]+)(?:\.([^\.\s]+))+")
    Dim m As Match = emailRegEx.Match(emailString)
    If m.Success Then
        Dim output As String = ""
        output &= "User name: " & m.Groups(1).Value & vbCrLf
        For i As Integer = 2 To m.Groups.Count - 1
            Dim g As Group = m.Groups(i)
            For Each c As Capture In g.Captures
                output &= "Address part: " & c.Value & vbCrLf
            Next
        Next
        MsgBox(output)
    Else
        MsgBox("The e-mail address cannot be parsed.")
    End If
End Sub
```

Cet exemple requiert que vous utilisiez l'instruction **Imports** pour importer l'espace de noms System.Text.RegularExpressions. Pour plus d'informations, consultez Imports, instruction.

Voir aussi

Tâches

Comment : vérifier que des chaînes sont dans un format d'adresse électronique valide

Autres ressources

Analyse de chaînes en Visual Basic

Comment : identifier les URL dans une chaîne en Visual Basic

Cet exemple montre comment utiliser une expression régulière simple pour identifier un URI (Uniform Resource Identifier) dans une chaîne. Pour limiter le nombre d'affirmations incorrectes (texte identifié par erreur comme un URI), un format URI spécifique est pris par défaut. Cela signifie que des négations incorrectes peuvent exister si des URI valides ne sont pas identifiés.

Exemple

Un format spécifique d'URI peut être mis en correspondance avec l'expression régulière `([^=":\s]+:)?//[^\s"]+` qui signifie :

1. Apparence facultative de :
    a. ensemble d'un ou de plusieurs caractères qui ne sont pas `=`, `"`, `:`, ou un caractère espace, suivi de ;
    b. le caractère `:`, suivi de ;
2. la chaîne `//`, suivie de ;
3. ensemble d'un ou de plusieurs caractères qui ne sont pas des guillemets ou des caractères espace.

L'objet Regex est initialisé avec l'expression régulière.

La méthode Matches de l'objet **Regex** retourne un objet MatchCollection qui contient des informations sur toutes les parties de la chaîne d'entrée correspondant à l'expression régulière.

```
''' <summary>Identifies URIs in text.</summary>
''' <param name="text">Text to parse.</param>
''' <remarks>Displays each URI in the input text.</remarks>
Sub IdentifyURIs(ByVal text As String)
    Dim uriRegex As New Regex("([^="":\s]+:)?//[^\s""]+")
    Dim output As String = ""
    For Each m As Match In uriRegex.Matches(text)
        output &= m.Value & vbCrLf
    Next
    MsgBox(output)
End Sub
```

Cet exemple requiert que vous utilisiez l'instruction **Imports** pour importer l'espace de noms System.Text.RegularExpressions. Pour plus d'informations, consultez Imports, instruction.

Voir aussi

Tâches

Comment : identifier des liens hypertexte dans une chaîne HTML en Visual Basic
Comment : supprimer des caractères non valides d'une chaîne

Autres ressources

Analyse de chaînes en Visual Basic

Comment : identifier le texte d'une chaîne HTML dans Visual Basic

Cet exemple montre comment utiliser une expression régulière simple pour supprimer les balises d'un document HTML.

Exemple

Les balises HTML peuvent correspondre à l'expression régulière `\<[^\>]+\>`, c'est-à-dire :

1. le caractère « < », suivi
2. d'un jeu d'un ou de plusieurs caractères, ne comprenant pas le « > », suivi
3. du caractère « > ».

Cet exemple utilise la méthode System.Text.RegularExpressions.Regex.Replace(System.String,System.String,System.String) partagée pour remplacer toutes les correspondances de l'expression régulière de la balise par la chaîne vide.

```
''' <summary>Removes the tags from an HTML document.</summary>
''' <param name="htmlText">HTML text to parse.</param>
''' <returns>The text of an HTML document without tags.</returns>
''' <remarks></remarks>
Function GetTextFromHtml(ByVal htmlText As String) As String
    Dim output As String = Regex.Replace(htmlText, "\<[^\>]+\>", "")
    Return output
End Function
```

Cet exemple requiert que vous utilisiez l'instruction **Imports** pour importer l'espace de noms System.Text.RegularExpressions. Pour plus d'informations, consultez Imports, instruction.

Voir aussi

Tâches

Comment : identifier des liens hypertexte dans une chaîne HTML en Visual Basic
Comment : supprimer des caractères non valides d'une chaîne

Autres ressources

Analyse de chaînes en Visual Basic

Comment : identifier des liens hypertexte dans une chaîne HTML en Visual Basic

Cet exemple présente une expression régulière simple pour identifier des liens hypertexte dans un document HTML.

Exemple

Cet exemple utilise l'expression régulière `<A[^>]*?HREF\s*=\s*"([^"]+)"[^>]*?>([\s\S]*?)<\/A>`, qui signifie :

1. la chaîne "<A", suivie de ;
2. le plus petit jeu de zéro ou plus de caractères qui n'inclut pas le caractère ">", suivi de ;
3. la chaîne "HREF", suivie de ;
4. zéro ou plus de caractères espace, suivis de ;
5. le caractère "=", suivi de ;
6. zéro ou plus de caractères espace, suivis de ;
7. le caractère guillemet, suivi de ;
8. le jeu de caractères qui n'inclut pas le caractère guillemet (capturé), suivi de ;
9. le caractère guillemet, suivi de ;
10. le plus petit jeu de zéro ou plus de caractères qui n'inclut pas le caractère ">", suivi de ;
11. le caractère ">", suivi de ;
12. le plus petit jeu de zéro ou plus de caractères (capturé), suivi de ;
13. la chaîne "</A>".

L'objet Regex est initialisé avec l'expression régulière, et spécifié comme respectant la casse.

La méthode Matches de l'objet **Regex** retourne un objet MatchCollection qui contient des informations sur toutes les parties de la chaîne d'entrée correspondant à l'expression régulière.

```
''' <summary>Identifies hyperlinks in HTML text.</summary>
''' <param name="htmlText">HTML text to parse.</param>
''' <remarks>This method displays the label and destination for
''' each link in the input text.</remarks>
Sub IdentifyLinks(ByVal htmlText As String)
    Dim hrefRegex As New Regex( _
        "<A[^>]*?HREF\s*=\s*""([^""]+)""[^>]*?>([\s\S]*?)<\/A>", _
        RegexOptions.IgnoreCase)
    Dim output As String = ""
    For Each m As Match In hrefRegex.Matches(htmlText)
        output &= "Link label: " & m.Groups(2).Value & vbCrLf
        output &= "Link destination: " & m.Groups(1).Value & vbCrLf
    Next
    MsgBox(output)
End Sub
```

Cet exemple requiert que vous utilisiez l'instruction **Imports** pour importer l'espace de noms System.Text.RegularExpressions. Pour plus d'informations, consultez Imports, instruction.

Voir aussi

Concepts

Exemple : recherche de valeurs HREF

Autres ressources

Analyse de chaînes en Visual Basic

Dépannage des chaînes en Visual Basic

Cette section fournit des conseils sur le dépannage de la manipulation de chaînes en Visual Basic.

Dans cette section

Dépannage des expressions régulières en Visual Basic

Répertorie les rubriques qui décrivent les concepts de base de l'utilisation de chaînes en Visual Basic.

Rubriques connexes

Nothing et les chaînes en Visual Basic

Explique comment le runtime Visual Basic et le .NET Framework évaluent **Nothing** différemment lorsqu'il s'agit de chaînes.

Comment les informations de culture affectent les chaînes dans Visual Basic

Explique comment Visual Basic utilise les informations sur la culture pour effectuer des conversions et des comparaisons de chaînes.

Accès à une chaîne de base zéro et accès à une chaîne de base un en Visual Basic

Compare comment Visual Basic et le .NET Framework fournissent un accès aux caractères d'une chaîne.

Voir aussi

Autres ressources

Chaînes en Visual Basic

Procédures pas à pas pour les chaînes en Visual Basic

Introduction aux chaînes dans Visual Basic

Dépannage des expressions régulières en Visual Basic

Cette rubrique présente des problèmes courants qui peuvent se produire lorsque vous utilisez des expressions régulières, fournit des conseils pour résoudre ces problèmes et indique une procédure pour accéder aux chaînes capturées.

Incompatibilité avec le modèle prévu

Vous trouverez ci-dessous des tâches courantes que vous pouvez exécuter avec les expressions régulières, ainsi que des conseils de dépannage si vous n'obtenez pas les résultats escomptés :

- **Validation de chaînes**. Une expression régulière de validation de chaîne doit commencer par le caractère **^**. Il indique au moteur de l'expression régulière de commencer la mise en correspondance du modèle spécifié au début de la chaîne. Pour plus d'informations, consultez Construction d'une fonction de validation en Visual Basic et Assertions atomiques de largeur nulle.
- **Mise en correspondance avec des quantificateurs.** Les quantificateurs d'expression régulière (*****, **+**, **?**, **{}**) sont *gourmands*, c'est-à-dire qu'ils établissent une correspondance avec la chaîne la plus longue possible. Toutefois, pour certains modèles, vous pouvez utiliser une correspondance *paresseuse* pour obtenir la chaîne la plus courte possible. Les quantificateurs d'expression régulière paresseux sont ***?**, **+?**, **??** et **{}?**. Pour plus d'informations, consultez Quantificateurs.
- **Mise en correspondance avec des quantificateurs imbriqués.** Lorsque vous utilisez des quantificateurs imbriqués, vérifiez que tous les quantificateurs sont soit gourmands soit paresseux. Sinon, la mise en correspondance peut produire des résultats inattendus.

Accès aux chaînes capturées

Une fois qu'une expression régulière a détecté vos chaînes prévues, vous pouvez capturer ces chaînes et accéder au contenu capturé. Vous pouvez utiliser des constructions de regroupement pour capturer les chaînes qui correspondent à des groupes de sous-expressions dans votre expression régulière. Pour plus d'informations, consultez Constructions de regroupement.

Plusieurs étapes permettent d'accéder aux chaînes capturées avec des groupes imbriqués.

Pour accéder à un texte capturé

1. Créez un objet Regex pour une expression régulière.
2. Appelez la méthode Match pour obtenir l'objet Match.

   L'objet **Match** contient des informations sur la correspondance de l'expression régulière avec la chaîne.
3. Parcourez les objets Group contenus dans la collection Groups de l'objet **Match**.

   L'objet **Group** contient des informations sur les résultats d'un seul groupe de capture.
4. Parcourez les objets Capture contenus dans la collection Captures de chaque objet **Group**.

   Chaque objet **Capture** contient des informations sur une seule sous-expression capturée, y compris la sous-chaîne et l'emplacement correspondants.

Par exemple, le code suivant montre comment accéder aux chaînes capturées avec une expression régulière qui contient trois groupes de capture :

```
    ''' <summary>
    ''' Parses an e-mail address into its parts.
    ''' </summary>
    ''' <param name="emailString">E-mail address to parse.</param>
    ''' <remarks> For example, this method displays the following
    ''' text when called with "someone@mail.contoso.com":
    ''' User name: someone
    ''' Address part: mail
    ''' Address part: contoso
    ''' Address part: com
    ''' </remarks>
    Sub ParseEmailAddress(ByVal emailString As String)
        Dim emailRegEx As New Regex("(\S+)@([^\.\s]+)(?:\.([^\.\s]+))+")
        Dim m As Match = emailRegEx.Match(emailString)
```

```
        If m.Success Then
            Dim output As String = ""
            output &= "User name: " & m.Groups(1).Value & vbCrLf
            For i As Integer = 2 To m.Groups.Count - 1
                Dim g As Group = m.Groups(i)
                For Each c As Capture In g.Captures
                    output &= "Address part: " & c.Value & vbCrLf
                Next
            Next
            MsgBox(output)
        Else
            MsgBox("The e-mail address cannot be parsed.")
        End If
    End Sub
```

Voir aussi

Tâches

Dépannage des chaînes en Visual Basic

Concepts

Construction d'une fonction de validation en Visual Basic

Constantes et énumérations en Visual Basic

Les constantes permettent d'utiliser des noms significatifs au lieu d'une valeur qui ne varie pas. Les constantes stockent des valeurs qui, comme leur nom l'indique, demeurent constantes lors de l'exécution d'une application. Vous pouvez utiliser des constantes pour rendre votre code plus lisible en remplaçant les nombres par des noms significatifs.

Les énumérations offrent un moyen pratique d'utiliser des ensembles de constantes connexes et d'associer des valeurs de constantes à des noms. Par exemple, vous pouvez déclarer une énumération pour un ensemble de constantes entières associées aux jours de la semaine, et utiliser ensuite les noms des jours dans le code plutôt que leurs valeurs entières.

Dans cette section

Constantes en Visual Basic

Les rubriques de cette section décrivent des constantes et leurs utilisations.

Vue d'ensemble des énumérations

Les rubriques de cette section décrivent des énumérations et leurs utilisations.

Constantes et énumérations intrinsèques

Décrit les constantes et les énumérations fournies par Visual Basic 2005.

Rubriques connexes

Const, instruction (Visual Basic)

Fournit la syntaxe et les règles d'utilisation pour l'instruction **Const**, utilisée pour déclarer des constantes.

Enum, instruction (Visual Basic)

Fournit la syntaxe et les règles d'utilisation pour l'instruction **Enum**, utilisée pour créer des énumérations.

Option Explicit, instruction (Visual Basic)

Fournit la syntaxe et les règles d'utilisation pour l'instruction **Option Explicit**, utilisée au niveau du module pour forcer la déclaration explicite de toutes les variables de ce module.

Option Strict, instruction

Fournit la syntaxe et les règles d'utilisation pour l'instruction **Option Strict**, qui limite les conversions de types de données implicites en conversions étendues uniquement.

Constantes en Visual Basic

Les constantes stockent des valeurs qui, comme leur nom l'indique, demeurent constantes lors de l'exécution d'une application. Utilisez des constantes dans votre code pour les valeurs utilisées à plusieurs reprises ou pour améliorer la lisibilité du code.

Dans cette section

Vue d'ensemble des constantes

Définit des constantes et indique où et quand les utiliser.

Constantes déclarées par Visual Basic

Répertorie les constantes prédéfinies par Visual Basic.

Comment : déclarer une constante

Explique comment utiliser l'instruction **Const** pour déclarer une constante et définir sa valeur ; la déclaration d'une constante permet d'assigner un nom significatif à la valeur.

Constantes définies par l'utilisateur

Décrit comment créer vos propres constantes, y compris des informations sur les portées et comment éviter les références circulaires.

Constantes et types de données littérales

Fournit des informations sur la façon dont le compilateur Visual Basic initialise des constantes lorsque **Option Explicit** est désactivé.

Comment : regrouper les valeurs de constante connexes

Illustre le groupement des valeurs de constante qui sont liées.

Référence

Const, instruction (Visual Basic)

Décrit l'instruction **Const** et son utilisation.

Option Strict, instruction

Décrit l'instruction **Option Strict** et son utilisation.

Rubriques connexes

Vue d'ensemble des énumérations

Décrit les énumérations et leur utilisation, y compris les tâches courantes impliquant des énumérations.

Constantes d'impression et d'affichage

Répertorie des constantes d'impression et d'affichage prédéfinies par Visual Basic.

Constantes et énumérations intrinsèques

Décrit les constantes et les énumérations fournies par Visual Basic.

Constantes et énumérations en Visual Basic

Modifications apportées aux constantes et aux énumérations dans la dernière version de Visual Basic.

Vue d'ensemble des constantes

Une constante est un nom significatif qui symbolise un nombre ou une chaîne invariable. Les constantes stockent des valeurs qui, comme leur nom l'indique, demeurent identiques lors de l'exécution d'une application. Vous pouvez améliorer considérablement la lisibilité de votre code et simplifier sa gestion en utilisant des constantes. Utilisez-les dans du code qui contient des valeurs qui réapparaissent ou qui dépend de certains nombres qui sont difficiles à mémoriser ou qui n'ont aucune signification évidente.

Comment créer et utiliser des constantes

Visual Basic contient plusieurs constantes prédéfinies, utilisées principalement pour l'impression et l'affichage. Vous pouvez également créer vos propres constantes avec l'instruction **Const**, en utilisant les mêmes indications que celles permettant de créer un nom de variable. Si **Option Strict** est **On**, vous devez déclarer le type de constante de manière explicite.

La portée d'une constante, qui correspond à l'ensemble du code qui peut y faire référence sans qualifier son nom, est identique à celle d'une variable déclarée au même emplacement. Pour créer une constante existant dans la portée d'une procédure particulière, vous devez la déclarer dans cette procédure. Pour créer une constante qui soit disponible pour l'ensemble d'une application, vous devez la déclarer à l'aide du mot clé **Public** dans la section des déclarations de la classe.

Remarque
Bien que les constantes ressemblent un peu aux variables, vous ne pouvez pas les modifier ni leur assigner de nouvelles valeurs, contrairement aux variables.

Les constantes utilisées dans votre code peuvent être définies par le modèle d'objet pour les contrôles ou les composants que vous utilisez ou être définies par l'utilisateur (celles que vous créez personnellement).

Constantes de compilation et d'exécution

Une constante de compilation est calculée lors de la compilation du code, alors qu'une constante d'exécution peut être calculée uniquement durant l'exécution de l'application. Une constante de compilation aura la même valeur chaque fois qu'une application s'exécute, tandis qu'une constante d'exécution peut changer à chaque fois. Les constantes de compilation sont nécessaires pour les cas, tels que les tailles de tableau, les expressions de cas ou les initialiseurs d'énumérateur.

Voir aussi

Tâches

Comment : déclarer une constante

Référence

Const, instruction (Visual Basic)

Public (Visual Basic)

Concepts

Constantes et types de données littérales

Constantes déclarées par Visual Basic

Autres ressources

Constantes et énumérations (Visual Basic)

Constantes en Visual Basic

Constantes déclarées par Visual Basic

Visual Basic fournit plusieurs constantes prédéfinies que les développeurs peuvent utiliser dans la compilation conditionnelle ainsi que pour l'impression et l'affichage des données.

Constantes de compilation conditionnelle

Le tableau suivant répertorie les constantes prédéfinies disponibles pour la compilation conditionnelle.

Constante	Description
CONFIG	Chaîne qui correspond au paramètre actuel de la zone **Configuration de la solution active** dans le **Gestionnaire de configurations**.
DEBUG	Valeur **Boolean** qui peut être définie dans la boîte de dialogue **Propriétés du projet**. Par défaut, la configuration de débogage pour un projet définit **DEBUG**. Lorsque **DEBUG** est défini, les méthodes de la classe Debug génèrent un résultat vers la fenêtre **Sortie**. Lorsque la constante n'est pas définie, les méthodes de classe **Debug** ne sont pas compilées et aucune sortie de débogage n'est générée.
TARGET	Chaîne représentant le type de sortie pour le projet ou le paramètre de l'option de ligne de commande **/target**. Les valeurs possibles de **TARGET** sont les suivantes : • « winexe » pour une application Windows ; • « exe » pour une application console ; • « library » pour une bibliothèque de classes ; • « module » pour un module. • L'option **/target** peut être définie dans l'environnement de développement intégré de Visual Studio. Pour plus d'informations, consultez /target (Visual Basic).
TRACE	Valeur **Boolean** qui peut être définie dans la boîte de dialogue **Propriétés du projet**. Par défaut, toutes les configurations pour un projet définissent **TRACE**. Lorsque **TRACE** est défini, les méthodes de la classe Trace génèrent un résultat vers la fenêtre **Sortie**. Lorsque la constante n'est pas définie, les méthodes de classe **Trace** ne sont pas compilées et aucune sortie **Trace** n'est générée.
VBC_VER	Nombre qui représente la version de Visual Basic, au format *major.minor*. Le numéro de version de Visual Basic 2005 est 8.0.

Constantes d'impression et d'affichage

Lorsque vous appelez des fonctions d'impression et d'affichage, vous pouvez utiliser les constantes suivantes dans votre code au lieu des valeurs réelles.

Constante	Description
vbCrLf	Combinaison de retour chariot et de saut de ligne
vbCr	Caractère de retour chariot
vbLf	Caractère de saut de ligne

vbNewLine	Caractère de saut de ligne
vbNullChar	Caractère ayant la valeur 0
vbNullString	Différent d'une chaîne de longueur nulle ("") ; utilisé pour l'appel de procédures externes
vbObjectError	Numéro de l'erreur. Les numéros d'erreur définis par l'utilisateur doivent être supérieurs à cette valeur. Par exemple : `Err.Raise(Number) = vbObjectError + 1000`
vbTab	Caractère de tabulation
vbBack	Caractère de retour arrière
vbFormFeed	Non utilisé dans Microsoft Windows
vbVerticalTab	Non utilisé dans Microsoft Windows

Voir aussi

Tâches

Comment : déclarer une constante

Concepts

Énumérations déclarées par Visual Basic
Vue d'ensemble des constantes
Constantes définies par l'utilisateur
Constantes et types de données littérales

Comment : déclarer une constante

L'instruction **Const** permet de déclarer une constante et de définir sa valeur. En déclarant une constante, vous assignez un nom significatif à une valeur. Une fois qu'une constante est déclarée, vous ne pouvez plus la modifier ou lui assigner une nouvelle valeur.

Vous déclarez une constante dans une procédure ou bien dans la section des déclarations d'un module, d'une classe ou d'une structure. Les constantes de niveau classe ou structure sont **Private** par défaut, mais elles peuvent également être déclarées comme **Public**, **Friend**, **Protected** ou **Protected Friend** pour le niveau approprié d'accès au code.

La constante doit avoir un nom symbolique valide (les règles sont identiques à celles permettant de créer des noms de variables) et une expression composée de constantes et d'opérateurs numériques ou de type chaîne (pas d'appels de fonction).

Remarque

Les options disponibles dans les boîtes de dialogue, ainsi que les noms et emplacements des commandes de menu que vous voyez, peuvent différer de ce qui est décrit dans l'aide selon vos paramètres actifs ou votre édition. Cette page d'aide est orientée sur les **Paramètres de développement généraux**. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour déclarer une constante

- Écrivez une déclaration qui inclut un spécificateur d'accès, le mot clé **Const** et une expression, comme illustré dans les exemples suivants :

  VB

```
Public Const DaysInYear = 365
Private Const WorkDays = 250
```

  Lorsque **Option Strict** est **On**, vous devez déclarer explicitement une constante en spécifiant un type de données (**Boolean**, **Byte**, **Char**, **DateTime**, **Decimal**, **Double**, **Integer**, **Long**, **Short**, **Single** ou **String**). Lorsque **Option Strict** est **Off**, le compilateur assigne un type de données à la constante. Pour plus d'informations, consultez Constantes et types de données littérales.

Pour déclarer une constante avec Option Strict On

- Avec **Option Strict On**, écrivez une déclaration qui inclut le mot clé **As** et un type de données explicite, comme illustré dans les exemples suivants :

  VB

```
Public Const MyInteger As Integer = 42
Private Const DaysInWeek As Short = 7
Protected Friend Const Funday As String = "Sunday"
```

  Vous pouvez déclarer plusieurs constantes sur une même ligne, mais il est recommandé de ne déclarer qu'une seule constante par ligne pour simplifier la lisibilité du code. Si vous déclarez plusieurs constantes sur une seule ligne, elles doivent toutes avoir le même niveau d'accès (**Public**, **Private**, **Friend**, **Protected** ou **Protected Friend**).

Pour déclarer plusieurs constantes sur une même ligne

- Séparez les déclarations par une virgule et un espace, comme dans l'exemple suivant :

```
Public Const Four As Integer = 4, Five As Integer = 5, Six As Integer = 44
```

Voir aussi

Tâches

Comment : déclarer des énumérations

Référence

Const, instruction (Visual Basic)

Option Strict, instruction

Concepts

Constantes et types de données littérales

Vue d'ensemble des énumérations

Vue d'ensemble des constantes

Énumérations et qualification de noms

Constantes et énumérations intrinsèques

Autres ressources

Constantes et énumérations en Visual Basic

Constantes définies par l'utilisateur

Une constante est un nom significatif qui symbolise un nombre ou une chaîne invariable. Les constantes stockent des valeurs qui, comme leur nom l'indique, demeurent constantes lors de l'exécution d'une application. Vous pouvez utiliser des constantes qui sont définies par les contrôles ou les composants que vous utilisez ; vous pouvez également créer vos propres constantes. Les constantes que vous créez sont appelées des constantes *définies par l'utilisateur*.

Vous déclarez une constante par l'instruction **Const**, en utilisant les mêmes indications que celles permettant de créer un nom de variable. Si **Option Strict** est **On**, vous devez déclarer le type de constante de manière explicite.

Utilisation de l'instruction Const

Une instruction **Const** peut représenter une valeur mathématique ou de date/heure :

VB

```
Const conPi = 3.14159265358979
Public Const conMaxPlanets As Integer = 9
Const conReleaseDate = #1/1/1995#
```

Elle peut également définir des constantes **String** :

VB

```
Public Const conVersion = "07.10.A"
Const conCodeName = "Enigma"
```

L'expression à la droite du signe égal (**=**) est souvent un nombre ou une chaîne littérale, mais elle peut également être une expression dont le résultat est un nombre ou une chaîne (même si cette expression ne peut pas contenir d'appels à des fonctions). Vous pouvez également définir des constantes en termes de constantes définies préalablement :

VB

```
Const conPi2 = conPi * 2
```

Portée des constantes définies par l'utilisateur

La portée d'une instruction **Const** est identique à celle d'une variable déclarée au même emplacement. Vous pouvez spécifier la portée de l'une des façons suivantes :

- Pour créer une constante existant uniquement dans une procédure, vous devez la déclarer dans cette procédure.
- Pour créer une constante qui soit disponible pour toutes les procédures d'une classe, mais pas pour le code en dehors du module, vous devez la déclarer dans la section des déclarations de la classe.
- Pour créer une constante qui soit disponible pour tous les membres d'un assembly, mais pas pour les clients externes de cet assembly, vous devez la déclarer à l'aide du mot clé **Friend** dans la section des déclarations de la classe.
- Pour créer une constante qui soit disponible pour l'ensemble de l'application, vous devez la déclarer à l'aide du mot clé **Public** dans la section des déclarations de la classe.

Pour plus d'informations, consultez Comment : déclarer une constante.

Comment éviter des références circulaires

Dans la mesure où les constantes peuvent être définies en termes d'autres constantes, le risque est de créer un *cycle* (ou une référence circulaire) par inadvertance entre deux constantes ou plus. Un cycle se produit en présence d'au moins deux constantes publiques (Public), chacune étant définie par rapport à l'autre, comme illustré dans cet exemple :

VB

```
    Public Const conA = conB * 2
...    Public Const conB = conA / 2
```

Lorsqu'un cycle se produit, Visual Basic génère une erreur de compilation.

Voir aussi

Tâches

Comment : déclarer des énumérations

Référence

Const, instruction (Visual Basic)

Option Strict, instruction

Concepts

Constantes et types de données littérales

Vue d'ensemble des énumérations

Vue d'ensemble des constantes

Énumérations et qualification de noms

Constantes et énumérations intrinsèques

Vue d'ensemble des constantes

Autres ressources

Constantes et énumérations en Visual Basic

Constantes et types de données littérales

Un littéral est une valeur qui est exprimée comme elle-même plutôt que comme la valeur d'une variable ou le résultat d'une expression, tel que le nombre 3 ou la chaîne "Hello". Une constante est un nom significatif qui prend la place d'un littéral et conserve cette même valeur partout dans le programme, par opposition à une variable dont la valeur peut se modifier.

Si **Option Strict** a la valeur **On**, toutes les constantes doivent être explicitement déclarées avec un type de données. Dans l'exemple suivant, le type de données de `MyByte` est déclaré explicitement comme type de données **Byte** :

```
Option Strict On
Public Const MyByte As Byte = 2
```

La déclaration explicite de types de données n'est pas ambiguë, et le code typé est plus simple à lire et à gérer que le code non typé. Toutefois, si **Option Strict** a la valeur **Off**, le compilateur utilise le type de l'expression qui a servi à initialiser la constante. Par défaut, un littéral entier numérique est casté en un type de données **Integer**. Le type de données par défaut des nombres à virgule flottante est **Double**, et les mots clés **True** et **False** spécifient une constante **Boolean**.

Littéraux et contrainte de type

Il arrive parfois que vous vouliez forcer un littéral en un type de données particulier ; par exemple, lors de l'assignation d'une très grande valeur littérale intégrale à une variable de type **Decimal**. L'exemple suivant génère une erreur :

```
Dim myDecimal as Decimal
myDecimal = 10000000000000000000   ' This causes a compiler error.
```

L'erreur provient de la représentation du littéral. Le type de données **Decimal** peut avoir une valeur de cette taille, mais le littéral est implicitement représenté comme un type **Long** qui ne peut pas contenir une telle valeur.

Vous pouvez forcer un littéral en un type de données particulier de deux façons : en lui ajoutant un caractère de type ou en le plaçant entre des caractères englobants. Un caractère de type ou des caractères englobants doivent immédiatement précéder et/ou suivre le littéral (aucun autre caractère ou espace ne doit venir s'interposer).

Pour que l'exemple précédent fonctionne, vous devez ajouter au littéral le caractère de type `D`, de sorte qu'il soit représenté en tant que type **Decimal** :

```
MyDecimal = 10000000000000000000D
```

L'exemple suivant illustre l'utilisation correcte des caractères de type et des caractères englobants :

```
Option Strict Off
Public Const DefaultInteger = 100   ' Default is Integer.
Public Const DefaultDouble = 54.3345612   ' Default is Double.
Public Const MyCharacter = "a"C   ' Forces constant to be a Char type.
Public Const MyDate = #01/15/01#   ' Demonstrates DateTime constants.
Public Const MyTime = #1:15:59 AM#
Public Const MyLong = 45L   ' Forces data type to be a Long.
Public Const MySingle = 45.55!   ' Forces data type to be a Single.
```

Le tableau suivant répertorie les caractères englobants et les caractères de type disponibles dans Visual Basic.

Type de données	Caractère englobant	Caractère de type ajouté
Boolean	(aucun)	(aucun)
Byte	(aucun)	(aucun)
Char	"	C
Date	#	(aucun)

Decimal	(aucun)	D ou @
Double	(aucun)	R ou #
Integer	(aucun)	I ou %
Long	(aucun)	L ou &
Short	(aucun)	S
Single	(aucun)	F ou !
String	"	(aucun)

Voir aussi

Tâches

Comment : déclarer une constante
Comment : déclarer des énumérations

Référence

Option Strict, instruction
Option Explicit, instruction (Visual Basic)
Liste des types de données (Visual Basic)

Concepts

Constantes définies par l'utilisateur
Vue d'ensemble des constantes
Vue d'ensemble des énumérations
Énumérations et qualification de noms
Constantes et énumérations intrinsèques

Comment : regrouper les valeurs de constante connexes

Une énumération est le meilleur moyen de regrouper des constantes connexes. L'ajout d'une instruction **Enum** dans la section des déclarations d'une classe ou d'un module permet de créer une énumération. Pour plus d'informations, consultez Comment : déclarer des énumérations.

Pour regrouper des valeurs de constante connexes

1. Écrivez une déclaration qui inclut un niveau d'accès de code, le mot clé **Enum** et un nom valide. Cet exemple crée l'énumération **Private**, `temperatureValues`.

   VB

   ```
   Private Enum temperatureValues
   ```

2. Définissez les constantes dans l'énumération. Cet exemple crée l'énumération **Public** `temperatureValues` et assigne ses valeurs.

   VB

   ```
   Public Enum temperatureValues
     Scorching
     Hot
     Lukewarm
     Cool
     Cold
   End Enum
   ```

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Concepts

Énumérations et qualification de noms
Quand utiliser une énumération
Constantes et énumérations intrinsèques
Vue d'ensemble des constantes
Constantes et types de données littérales

Vue d'ensemble des énumérations

Les énumérations offrent un moyen pratique d'utiliser des ensembles de constantes connexes et d'associer des valeurs de constantes à des noms. Par exemple, vous pouvez déclarer une énumération pour un ensemble de constantes entières associées aux jours de la semaine, et utiliser ensuite les noms des jours dans le code plutôt que leurs valeurs entières.

Tâches impliquant des énumérations

Le tableau suivant répertorie des tâches courantes qui impliquent des énumérations.

Pour	Reportez-vous à
Rechercher une énumération prédéfinie	Énumérations déclarées par Visual Basic
Déclarer une énumération	Comment : déclarer des énumérations
Qualifier complètement le nom d'une énumération	Énumérations et qualification de noms
Faire référence à un membre d'énumération	Comment : faire référence à un membre d'énumération
Parcourir une énumération	Comment : itérer sur une énumération dans Visual Basic
Déterminer la chaîne associée à une énumération	Comment : déterminer la chaîne associée à une valeur d'énumération
Déterminer quand utiliser une énumération	Quand utiliser une énumération

Voir aussi

Tâches

Comment : déclarer une constante

Référence

Enum, instruction (Visual Basic)

Concepts

Vue d'ensemble des constantes

Constantes définies par l'utilisateur

Constantes et types de données littérales

Constantes et énumérations intrinsèques

Énumérations déclarées par Visual Basic

Les énumérations offrent un moyen pratique d'utiliser des ensembles de constantes connexes et d'associer des valeurs de constantes à des noms. Visual Basic fournit plusieurs énumérations utiles.

Énumérations prédéfinies

Le tableau suivant répertorie et décrit les énumérations fournies par Visual Basic.

Énumération	Description
AppWinStyle, énumération	Indique le style de fenêtre à utiliser pour le programme appelé lors de l'appel de la fonction **Shell**.
AudioPlayMode, énumération	Indique comment lire des sons lors de l'appel de méthodes audio.
BuiltInRole, énumération	Indique le type de rôle à vérifier lors de l'appel à la méthode My.User.IsInRole.
CallType, énumération	Indique le type de la procédure qui est appelée lors de l'appel à la fonction **CallByName**.
CompareMethod, énumération	Indique comment comparer des chaînes lors de l'appel de fonctions de comparaison.
DateFormat, énumération	Indique comment afficher les dates lors de l'appel de la fonction **FormatDateTime**.
DateInterval, énumération	Indique comment déterminer et mettre en forme des intervalles de date lors de l'appel de fonctions liées aux dates.
DeleteDirectoryOption, énumération	Spécifie l'opération à effectuer lorsqu'un répertoire à supprimer contient des fichiers ou des répertoires.
DueDate, énumération	Indique la date d'échéance des paiements lors de l'appel à des méthodes financières.
FieldType, énumération	Indique si les champs de texte sont délimités ou à largeur fixe.
FileAttribute, énumération	Indique les attributs de fichier à utiliser lors de l'appel de fonctions d'accès aux fichiers.
FirstDayOfWeek, énumération	Indique le premier jour de la semaine à utiliser lors de l'appel de fonctions liées aux dates.
FirstWeekOfYear, énumération	Indique la première semaine de l'année à utiliser lors de l'appel de fonctions liées aux dates.
MsgBoxResult, énumération	Indique le bouton sur lequel vous avez appuyé dans un message, retourné par la fonction **MsgBox**.
MsgBoxStyle, énumération	Indique les boutons à afficher lors de l'appel de la fonction **MsgBox**.
OpenAccess, énumération	Indique le type d'accès lors de l'appel de fonctions d'accès aux fichiers.
OpenMode, énumération	Indique le mode à utiliser lors de l'appel de fonctions d'accès aux fichiers.
OpenShare, énumération	Indique si un fichier peut être utilisé par d'autres processus lors de l'appel de fonctions d'accès aux fichiers.
RecycleOption, énumération	Spécifie si un fichier doit être supprimé définitivement ou placé dans la **Corbeille**.

SearchOption, énumération	Spécifie si la recherche doit être effectuée dans tous les répertoires ou uniquement ceux de niveau supérieur.
Tristate, énumération	Indique une valeur **Boolean** ou signale que la valeur par défaut doit être utilisée lors de l'appel de fonctions de mise en forme de nombre.
UICancelOption, énumération	Spécifie l'action à effectuer si l'utilisateur clique sur **Annuler** pendant une opération.
UIOption, énumération	Spécifie si une boîte de dialogue de progression doit s'afficher ou non lors de la copie, la suppression ou le déplacement de fichiers ou de répertoires.
VariantType, énumération	Indique le type d'un objet variant, retourné par la fonction **VarType**.
VbStrConv, énumération	Indique le type de conversion à effectuer lors de l'appel de la fonction **StrConv**.
DueDate, énumération	Indique la date d'échéance des paiements lors de l'appel de méthodes financières.

Voir aussi

Tâches

Comment : déclarer des énumérations
Comment : faire référence à un membre d'énumération
Comment : itérer sur une énumération dans Visual Basic
Comment : déterminer la chaîne associée à une valeur d'énumération

Concepts

Énumérations et qualification de noms
Quand utiliser une énumération

Comment : déclarer des énumérations

L'ajout d'une instruction **Enum** dans la section des déclarations d'une classe ou d'un module permet de créer une énumération. Vous ne pouvez pas déclarer d'énumération dans une méthode. Pour spécifier le niveau d'accès approprié, utilisez **Private**, **Protected**, **Friend** ou **Public**.

Un type **Enum** possède un nom, un type sous-jacent et un ensemble de champs, dont chacun représente une constante. Le nom doit être un qualificateur Visual Basic 2005 valide. Le type sous-jacent doit être l'un des types d'entiers : **Byte**, **Short**, **Long** ou **Integer**. **Integer** est le type par défaut. Les énumérations sont toujours fortement typées et ne sont pas interchangeables avec les types de nombre entier.

Les énumérations ne peuvent pas comporter de valeurs à virgule flottante. Si une valeur à virgule flottante avec **Option Strict On** est assignée à une énumération, une erreur de compilation se produit. Si **Option Strict** est **Off**, la valeur est convertie automatiquement en type **Enum**.

Pour plus d'informations sur les noms et sur l'utilisation de l'instruction **Imports** pour rendre inutile la qualification de noms, consultez Énumérations et qualification de noms.

Pour déclarer une énumération

1. Écrivez une déclaration qui inclut un niveau d'accès de code, le mot clé **Enum** et un nom valide, comme dans les exemples suivants, qui déclarent chacun un **Enum** différent.

   VB

   ```
   Private Enum SampleEnum
           SampleMember
       End Enum
       Public Enum SampleEnum2
           SampleMember
       End Enum
       Protected Enum SampleEnum3
           SampleMember
       End Enum
       Friend Enum SampleEnum4
           SampleMember
       End Enum
       Protected Friend Enum SampleEnum5
           SampleMember
       End Enum
   ```

2. Définissez les constantes dans l'énumération. Par défaut, la première constante d'une énumération est initialisée à la valeur `0` et les constantes suivantes sont initialisées à une valeur de plus que la constante précédente. Par exemple, l'énumération suivante (`Days`) contient une constante nommée `Sunday` avec la valeur `0`, une constante nommée `Monday` avec la valeur `1`, une constante nommée `Tuesday` avec la valeur `2`, et ainsi de suite.

   VB

   ```
   Public Enum Days
     Sunday
     Monday
     Tuesday
     Wednesday
     Thursday
     Friday
     Saturday
   End Enum
   ```

3. Vous pouvez assigner explicitement des valeurs à des constantes dans une énumération en utilisant une instruction d'assignation. Vous pouvez assigner une valeur entière, y compris des nombres négatifs. Par exemple, vous pouvez définir des constantes avec des valeurs inférieures à zéro pour représenter des conditions d'erreur. Dans l'énumération suivante, la valeur `-1` est explicitement assignée à la constante `Invalid` et la valeur `0` est assignée à la constante `Sunday`. Dans la mesure où `Saturday` est la première constante de l'énumération, la valeur `0` lui est également assignée. La valeur de `Monday` est `1` (un de plus que la valeur de `Sunday`), la valeur de `Tuesday` est `2`, et ainsi de suite.

   VB

   ```
   Public Enum WorkDays
     Saturday
     Sunday = 0
     Monday
     Tuesday
     Wednesday
     Thursday
     Friday
     Invalid = -1
   End Enum
   ```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Pour déclarer explicitement une énumération

- Écrivez une déclaration en utilisant la syntaxe suivante.

  VB

  ```
  Public Enum MyEnum As Byte
    Zero
    One
    Two
  End Enum
  ```

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération
Comment : itérer sur une énumération dans Visual Basic
Comment : déterminer la chaîne associée à une valeur d'énumération

Concepts

Énumérations et qualification de noms
Quand utiliser une énumération
Constantes et énumérations intrinsèques
Constantes et types de données littérales

Autres ressources

Constantes en Visual Basic

Comment : faire référence à un membre d'énumération

Les énumérations offrent un moyen pratique d'utiliser des ensembles de constantes connexes et d'associer des valeurs de constantes à des noms. Par exemple, vous pouvez déclarer une énumération pour un ensemble de constantes entières associées aux jours de la semaine, et utiliser ensuite les noms des jours dans le code plutôt que leurs valeurs entières.

Vous pouvez éviter d'utiliser des noms qualifiés complets avec l'instruction **Imports**. Pour plus d'informations, consultez Énumérations et qualification de noms.

Pour faire référence à un membre d'énumération

- Qualifiez le nom de membre avec l'énumération. Par exemple, l'exemple suivant assigne le membre **Saturday** de l'énumération **FirstDayOfWeek** à la variable `DayValue`.

  VB

```
DayValue = FirstDayOfWeek.Saturday
```

Voir aussi

Tâches

Comment : déclarer des énumérations
Comment : itérer sur une énumération dans Visual Basic
Comment : déterminer la chaîne associée à une valeur d'énumération

Concepts

Énumérations déclarées par Visual Basic
Énumérations et qualification de noms
Quand utiliser une énumération

Énumérations et qualification de noms

Lorsque vous faites référence à un membre d'une énumération, vous devez généralement le désigner par le nom de votre énumération. Par exemple, pour faire référence au membre `Sunday` de votre énumération `Days`, vous devez utiliser la syntaxe suivante :

VB

```
X = Days.Sunday
```

Utilisation de l'instruction Imports

Vous pouvez éviter d'employer le nom complet en ajoutant une instruction **Imports** dans la section des déclarations d'espace de noms de votre code, comme dans l'exemple suivant :

VB

```
Imports WindowsApplication1.Form1.Days
Imports WindowsApplication1.Form1.WorkDays
```

Une instruction **Imports** importe des noms d'espaces de noms à partir de projets et d'assemblys référencés et à partir de noms définis dans le même projet que le module dans lequel figure l'instruction. Dès que cette instruction est ajoutée, vous pouvez faire référence aux membres de l'énumération sans qualification, comme dans l'exemple suivant :

VB

```
X = Sunday
```

Si vous organisez des ensembles de constantes connexes dans des énumérations, vous pouvez utiliser les mêmes noms de constantes dans des contextes distincts. Par exemple, vous pouvez utiliser des noms identiques pour les constantes des jours de la semaine dans les énumérations `Days` et `WorkDays`. Si vous utilisez l'instruction **Imports** avec vos énumérations, vous devez éviter d'utiliser des références ambiguës. Prenons l'exemple suivant :

VB

```
Imports WindowsApplication1.Form1.Days
Imports WindowsApplication1.Form1.WorkDays
```

VB

```
Public Sub New()
  ' Insert code to implement constructor.
  X = Monday
End Sub
```

Si `Monday` est un membre des énumérations `Days` et `Workdays`, ce code génère une erreur de compilation. Pour éviter les références ambiguës lorsque vous faites référence à une constante individuelle, vous devez désigner le nom de la constante par son énumération. Le code suivant fait référence aux constantes `Saturday` dans les énumérations `Days` et `WorkDays`.

VB

```
X = Days.Saturday
Y = WorkDays.Saturday
```

Voir aussi

Tâches

Comment : déclarer des énumérations

Comment : faire référence à un membre d'énumération
Comment : itérer sur une énumération dans Visual Basic
Comment : déterminer la chaîne associée à une valeur d'énumération

Référence

Enum, instruction (Visual Basic)
Imports, instruction
Liste des types de données (Visual Basic)

Concepts

Énumérations déclarées par Visual Basic
Quand utiliser une énumération
Constantes et types de données littérales

Comment : itérer sur une énumération dans Visual Basic

Les énumérations offrent un moyen pratique d'utiliser des ensembles de constantes connexes et d'associer des valeurs de constantes à des noms. Pour parcourir une énumération, vous pouvez la déplacer dans un tableau à l'aide de la méthode GetValues. Vous pourriez également parcourir une énumération à l'aide d'une instruction **For...Each**, en utilisant la méthode GetNames ou **GetValues** pour extraire la chaîne ou valeur numérique.

Pour parcourir une énumération

- Déclarez un tableau et convertissez l'énumération en ce tableau avec la méthode **GetValues**, avant de passer le tableau comme vous le feriez avec toute autre variable. L'exemple suivant affiche chaque membre de l'énumération `MyEnum` qui itère sur l'énumération.

  VB

```
Dim items As Array
items = System.Enum.GetValues(GetType(FirstDayOfWeek))
Dim item As String
For Each item In items
    MsgBox(item)
Next
```

Voir aussi

Tâches

Comment : déclarer des énumérations
Comment : déterminer la chaîne associée à une valeur d'énumération
Comment : faire référence à un membre d'énumération
Comment : passer un tableau à une procédure ou propriété

Concepts

Vue d'ensemble des énumérations
Quand utiliser une énumération
Énumérations et qualification de noms
Énumérations déclarées par Visual Basic
Vue d'ensemble des tableaux dans Visual Basic

Comment : déterminer la chaîne associée à une valeur d'énumération

Les méthodes GetValues et GetNames vous permettent de déterminer les chaînes et valeurs associées à des membres d'énumération.

Pour déterminer la chaîne associée à une énumération

- Utilisez la méthode **GetNames** pour récupérer les chaînes associées aux membres d'énumération. Cet exemple déclare une énumération, `flavorEnum`, puis utilise la méthode **GetNames** pour afficher les chaînes associées à chaque membre.

  VB

```
Public Enum flavorEnum
  salty
  sweet
  sour
  bitter
End Enum

Private Sub TestMethod()
  MsgBox("The strings in the flavorEnum are:")
  Dim i As String
  For Each i In [Enum].GetNames(GetType(flavorEnum))
      MsgBox(i)
  Next
End Sub
```

Voir aussi

Tâches

Comment : déclarer des énumérations
Comment : faire référence à un membre d'énumération
Comment : itérer sur une énumération dans Visual Basic

Référence

Enum, instruction (Visual Basic)
GetValues
GetNames
Enum

Concepts

Énumérations déclarées par Visual Basic
Énumérations et qualification de noms
Quand utiliser une énumération

Quand utiliser une énumération

Les énumérations facilitent l'utilisation des ensembles de constantes connexes. Une énumération, ou **Enum**, est un nom symbolique pour un ensemble de valeurs. Les énumérations sont traitées comme des types de données et vous pouvez les utiliser pour créer des jeux de constantes utilisables avec des variables et des propriétés.

Quand utiliser une énumération

À chaque fois qu'une procédure accepte un jeu limité de variables, envisagez d'utiliser une énumération. Les énumérations rendent le code plus propre et plus lisible, en particulier lorsque des noms explicites sont utilisés.

Les avantages liés à l'utilisation d'énumérations sont les suivants :

- elle permet de réduire les erreurs dues à la transposition ou à l'entrée incorrecte de nombres ;
- elle facilite la modification ultérieure des valeurs ;
- elle augmente la lisibilité du code, ce qui réduit le risque d'erreurs ;
- elle garantit la compatibilité ascendante. Avec les énumérations, votre code est moins susceptible d'échouer si dans l'avenir quelqu'un modifie les valeurs correspondent aux noms de membres.

Affectation de noms aux énumérations

Utilisez une convention d'affectation de noms pour les membres d'énumération. Lorsque Visual Basic rencontre un nom de membre d'énumération, une exception peut être levée si d'autres bibliothèques de types référencées contiennent le même nom. Utilisez un préfixe unique qui identifie les valeurs de votre application ou composant.

Lorsque vous faites référence à un membre d'une énumération, vous devez qualifier le nom du membre avec le nom d'énumération ou utiliser l'instruction **Imports**. Pour plus d'informations, consultez Énumérations et qualification de noms.

Énumérations prédéfinies

Visual Basic fournit plusieurs énumérations prédéfinies, telles que **FirstDayOfWeek** et **MsgBoxResul**t, pour faciliter l'écriture de code. Pour obtenir une liste de ces énumérations, consultez Énumérations déclarées par Visual Basic.

Voir aussi

Tâches

Comment : déclarer des énumérations
Comment : faire référence à un membre d'énumération
Comment : itérer sur une énumération dans Visual Basic
Comment : déterminer la chaîne associée à une valeur d'énumération

Référence

Enum, instruction (Visual Basic)

Concepts

Énumérations et qualification de noms
Constantes et énumérations intrinsèques

Constantes et énumérations intrinsèques

Visual Basic propose de nombreuses constantes et énumérations intrinsèques pour faciliter la programmation.

Notes

Les constantes stockent des valeurs qui demeurent constantes lors de l'exécution d'une application. Ce sont des noms significatifs qui remplacent un nombre ou une chaîne et facilitent la lecture du code. Les énumérations facilitent l'utilisation des ensembles de constantes connexes. Une énumération, ou Enum, est un nom symbolique pour un ensemble de valeurs.

Vous pouvez créer vos propres constantes et énumérations, comme indiqué dans Constantes définies par l'utilisateur, ou utiliser les constantes et énumérations intrinsèques fournies avec Visual Basic, qui sont répertoriées dans l'Explorateur d'objets.

Le tableau suivant répertorie et décrit les énumérations intrinsèques disponibles dans Visual Basic.

Énumération	Description
AppWinStyle, énumération	Indique le style de fenêtre à utiliser pour le programme appelé lors de l'appel de la fonction **Shell**.
AudioPlayMode, énumération	Indique comment lire des sons lors de l'appel de méthodes audio.
BuiltInRole, énumération	Indique le type de rôle à vérifier lors de l'appel à la méthode My.User.IsInRole.
CallType, énumération	Indique le type de la procédure qui est appelée lors de l'appel de la fonction **CallByName**.
CompareMethod, énumération	Indique comment comparer des chaînes lors de l'appel de fonctions de comparaison.
DateFormat, énumération	Indique comment afficher les dates lors de l'appel de la fonction **FormatDateTime**.
DateInterval, énumération	Indique comment déterminer et mettre en forme des intervalles de date lors de l'appel de fonctions liées aux dates.
DeleteDirectoryOption, énumération	Spécifie l'opération à effectuer lorsqu'un répertoire à supprimer contient des fichiers ou des répertoires.
DueDate, énumération	Indique la date d'échéance des paiements lors de l'appel de méthodes financières.
FieldType, énumération	Indique si les champs de texte sont délimités ou à largeur fixe.
FileAttribute, énumération	Indique les attributs de fichier à utiliser lors de l'appel de fonctions d'accès aux fichiers.
FirstDayOfWeek, énumération	Indique le premier jour de la semaine à utiliser lors de l'appel de fonctions liées aux dates.
FirstWeekOfYear, énumération	Indique la première semaine de l'année à utiliser lors de l'appel de fonctions liées aux dates.
MsgBoxResult, énumération	Indique le bouton sur lequel vous avez appuyé dans un message, retourné par la fonction **MsgBox**.
MsgBoxStyle, énumération	Indique les boutons à afficher lors de l'appel de la fonction **MsgBox**.
OpenAccess, énumération	Identifie si un fichier est accessible en lecture, en écriture ou les deux.
OpenMode, énumération	Indique quel mode d'accès doit être utilisé lors de l'accès à un fichier.

OpenShare, énumération	Spécifie si d'autres processus peuvent accéder à un fichier ouvert.
RecycleOption, énumération	Spécifie si un fichier doit être supprimé définitivement ou être placé dans la **Corbeille**.
SearchOption, énumération	Spécifie s'il faut rechercher tous les répertoires ou uniquement les répertoires de niveau supérieur.
Tristate, énumération	Indique une valeur Boolean ou si la valeur par défaut doit être utilisée lors de l'appel de fonctions de mise en forme de nombres.
UICancelOption, énumération	Spécifie l'opération à effectuer si l'utilisateur clique sur **Annuler** pendant une opération.
UIOption, énumération	Spécifie l'affichage ou non d'une boîte de dialogue de progression lors de la copie, de la suppression ou du déplacement de fichiers ou de répertoires.
VariantType, énumération	Indique le type d'un objet variant, retourné par la fonction **VarType**.
VbStrConv, énumération	Indique le type de conversion à effectuer lors de l'appel de la fonction **StrConv**.

Voir aussi

Tâches

Comment : déclarer une constante

Référence

Enum, instruction (Visual Basic)

Explorateur d'objets

Concepts

Vue d'ensemble des constantes

Constantes et types de données littérales

Vue d'ensemble des énumérations

Constantes définies par l'utilisateur

Autres ressources

Constantes et énumérations (Visual Basic)

Opérateurs et expressions en Visual Basic

Un *opérateur* est un élément de code qui exécute une opération sur un ou plusieurs éléments de code qui contiennent des valeurs. Les éléments de valeur comprennent des variables, des constantes, des littéraux, des propriétés, des retours provenant des procédures **Function** et **Operator** et des expressions.

Une *expression* est une série d'éléments de valeur associés aux opérateurs, ce qui retourne une nouvelle valeur. Les opérateurs agissent sur les éléments de valeur en effectuant des calculs, des comparaisons et d'autres opérations.

Types d'opérateurs

Visual Basic fournit les types d'opérateurs suivants :

- Les opérateurs arithmétiques qui exécutent des calculs familiers sur les valeurs numériques, y compris le décalage de leurs modèles binaires.
- Les opérateurs de comparaison qui comparent deux expressions et retournent une valeur **Boolean** représentant le résultat de la comparaison.
- Les opérateurs de concaténation qui relient plusieurs chaînes dans une seule chaîne.
- Les opérateurs logiques et au niveau du bit en Visual Basic qui combinent des valeurs **Boolean** ou numériques et retournent un résultat du même type de données que les valeurs.

Les éléments de valeur associés à un opérateur sont appelés *opérandes* de cet opérateur. Les opérateurs associés aux éléments de valeur forment des expressions, à l'exception de l'opérateur d'assignation qui forme une *instruction*. Pour plus d'informations, consultez Instructions d'assignation.

Évaluation des expressions

Le résultat final d'une expression représente une valeur qui fait partie généralement d'un type de données familier tel que **Boolean**, **String** ou d'un type numérique.

Voici des exemples d'expressions.

```
5 + 4

' The preceding expression evaluates to 9.

15 * System.Math.Sqrt(9) + x

' The preceding expression evaluates to 45 plus the value of x.

"Concat" & "ena" & "tion"

' The preceding expression evaluates to "Concatenation".

763 < 23

' The preceding expression evaluates to False.
```

Plusieurs opérateurs peuvent effectuer des actions dans une expression ou une instruction unique, comme le montrent les exemples ci-dessous.

VB

```
x = 45 + y * z ^ 2
```

Dans l'exemple précédent, Visual Basic exécute les opérations de l'expression à droite de l'opérateur d'assignation (**=**), puis assigne la valeur résultante à la variable `x` située à gauche. Le nombre d'opérateurs pouvant être combinés dans une expression est illimité, mais la connaissance de Priorité des opérateurs en Visual Basic est nécessaire pour garantir que vous obtiendrez les résultats escomptés.

Voir aussi

Concepts

Association efficace d'opérateurs

Autres ressources

Opérateurs (Visual Basic)

Instructions dans Visual Basic

Tâches courantes effectuées avec les opérateurs de Visual Basic

Les opérateurs exécutent de nombreuses de tâches courantes qui impliquent une ou plusieurs expressions appelées *opérandes*.

Tâches arithmétiques et de décalage de bits

Le tableau suivant récapitule les opérations arithmétiques et de décalage de bits disponibles.

Pour	Reportez-vous à
Ajouter une valeur numérique à une autre	+, opérateur (Visual Basic)
Soustraire une valeur numérique d'une autre	-, opérateur (Visual Basic)
Inverser le signe d'une valeur numérique	-, opérateur (Visual Basic)
Multiplier une valeur numérique par une autre	*, opérateur (Visual Basic)
Diviser une valeur numérique par une autre	/, opérateur (Visual Basic)
Rechercher le quotient d'une valeur numérique divisée par une autre (sans le reste)	\, opérateur
Rechercher le reste d'une valeur numérique divisée par une autre (sans le quotient)	Mod, opérateur (Visual Basic)
Élever une valeur numérique à la puissance d'une autre	^, opérateur (Visual Basic)
Déplacer à gauche le modèle binaire d'une valeur numérique	< < opérateur (Visual Basic)
Déplacer à droite le modèle binaire d'une valeur numérique	> >, opérateur (Visual Basic)

Tâches de comparaison

Le tableau suivant récapitule les opérations de comparaison disponibles :

Pour	Reportez-vous à
Déterminer si deux valeurs sont égales	Opérateur **=** (Opérateurs de comparaison en Visual Basic)
Déterminer si deux valeurs sont inégales	Opérateur **<>** (Opérateurs de comparaison en Visual Basic)
Déterminer si une valeur est inférieure à une autre	Opérateur **<** (Opérateurs de comparaison en Visual Basic)
Déterminer si une valeur est supérieure à une autre	Opérateur **>** (Opérateurs de comparaison en Visual Basic)
Déterminer si une valeur est inférieure ou égale à une autre	Opérateur **<=** (Opérateurs de comparaison en Visual Basic)
Déterminer si une valeur est supérieure ou égale à une autre	Opérateur **>=** (Opérateurs de comparaison en Visual Basic)

Déterminer si deux variables objets font référence à la même instance de l'objet	Is, opérateur (Visual Basic)
Déterminer si deux variables objets font référence à différentes instances de l'objet	IsNot, opérateur
Déterminer si un objet est d'un type spécifique	TypeOf, opérateur (Visual Basic)

Tâches de concaténation

Le tableau suivant récapitule les opérations de concaténation disponibles :

Pour	**Reportez-vous à**
Joindre plusieurs chaînes dans une chaîne unique	Opérateur **&** (Opérateurs de concaténation (Visual Basic))
Joindre des valeurs numériques avec les valeurs de chaîne	Opérateur **+** (Opérateurs de concaténation (Visual Basic))

Tâches logiques et au niveau du bit

Le tableau suivant récapitule les opérations logiques et au niveau du bit disponibles.

Pour	**Reportez-vous à**
Effectuer une négation logique sur une valeur Boolean	Not, opérateur (Visual Basic)
Effectuer une conjonction logique sur deux valeurs Boolean	And, opérateur (Visual Basic)
Effectuer une disjonction logique inclusive sur deux valeurs Boolean	Or, opérateur (Visual Basic)
Effectuer une disjonction logique exclusive sur deux valeurs Boolean	Xor, opérateur (Visual Basic)
Effectuer une conjonction logique court-circuitée sur deux valeurs Boolean	AndAlso, opérateur
Effectuer une disjonction logique inclusive court-circuitée sur deux valeurs Boolean	OrElse, opérateur
Effectuer une conjonction logique bit par bit sur deux valeurs intégrales	And, opérateur (Visual Basic)
Effectuer une disjonction logique inclusive bit par bit sur deux valeurs intégrales	Or, opérateur (Visual Basic)
Effectuer une disjonction logique exclusive bit par bit sur deux valeurs intégrales	Xor, opérateur (Visual Basic)
Effectuer une négation logique bit par bit sur une valeur intégrale	Not, opérateur (Visual Basic)

Voir aussi

Référence

Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs et expressions en Visual Basic

Opérateurs arithmétiques en Visual Basic

Les opérateurs arithmétiques permettent d'effectuer de nombreuses opérations arithmétiques familières qui impliquent le calcul de valeurs numériques représentées par des littéraux, des variables, d'autres expressions, des appels de fonction et de propriété et des constantes. Également classés avec les opérateurs arithmétiques figurent les opérateurs de décalage de bits, qui agissent au niveau des bits des opérandes et décalent leurs modèles binaires vers la gauche ou vers la droite.

Opérations arithmétiques

Vous pouvez ajouter deux valeurs dans une expression à l'aide de l'+, opérateur (Visual Basic), ou soustraire une valeur d'une autre à l'aide de l'-, opérateur (Visual Basic), comme le montre l'exemple ci-dessous.

VB

```
Dim x As Integer
x = 67 + 34
x = 32 - 12
```

La négation utilise également l'-, opérateur (Visual Basic), mais avec un seul opérande, comme le montre l'exemple ci-dessous.

VB

```
Dim x As Integer = 65
Dim y As Integer
y = -x
```

La multiplication et la division utilisent l'*, opérateur (Visual Basic) et l'/, opérateur (Visual Basic), respectivement, comme le montre l'exemple ci-dessous.

VB

```
Dim y As Double
y = 45 * 55.23
y = 32 / 23
```

L'élévation à la puissance utilise l'^, opérateur (Visual Basic), comme le montre l'exemple ci-dessous.

VB

```
Dim z As Double
z = 23 ^ 3
' The preceding statement sets z to 12167 (the cube of 23).
```

La division par un entier s'effectue à l'aide de l'\, opérateur. Elle retourne le quotient, c'est-à-dire l'entier qui représente le nombre de fois que le diviseur peut être divisé par le dividende sans tenir compte du reste. Le diviseur et le dividende doivent être des types intégraux (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** et **ULong**) pour cet opérateur. Tous les autres types doivent d'abord être convertis en un type intégral. L'exemple suivant illustre la division d'entier.

VB

```
Dim k As Integer
k = 23 \ 5
' The preceding statement sets k to 4.
```

Le modulo arithmétique est effectué par l'Mod, opérateur (Visual Basic). Cet opérateur retourne le reste après que le diviseur est divisé par le dividende un nombre intégral de fois. Si le diviseur et le dividende sont des types intégraux, la valeur retournée est un intégral. Si le diviseur et le dividende sont des types à virgule flottante, la valeur retournée est également à virgule flottante. L'exemple suivant illustre ce comportement.

VB

```
Dim x As Integer = 100
Dim y As Integer = 6
Dim z As Integer
z = x Mod y
' The preceding statement sets z to 4.
```

VB

```
Dim a As Double = 100.3
Dim b As Double = 4.13
Dim c As Double
c = a Mod b
' The preceding statement sets c to 1.18.
```

Essai de division par zéro

La division par zéro produit divers résultats selon les types de données impliquées. Dans les divisions intégrales (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long**, **ULong**), le .NET Framework lève une exception DivideByZeroException. Dans les opérations de division sur le type de données **Decimal** ou **Single**, le .NET Framework lève également une exception **DivideByZeroException**.

Dans les divisions à virgule flottante qui impliquent le type de données **Double**, aucune exception n'est levée, et le résultat est le membre de la classe qui représente NaN, PositiveInfinity ou NegativeInfinity, selon le dividende. Le tableau suivant résume les divers résultats lors d'une tentative de division d'une valeur **Double** par zéro.

Type de données d'un dividende	Type de données d'un diviseur	Valeur du dividende	Résultat
Double	**Double**	0	**NaN** (pas un nombre défini mathématiquement)
Double	**Double**	> 0	**PositiveInfinity**
Double	**Double**	< 0	**NegativeInfinity**

Lorsque vous interceptez une exception **DivideByZeroException**, vous pouvez utiliser ses membres pour faciliter sa gestion. Par exemple, la propriété Message gère le texte du message pour l'exception. Pour plus d'informations, consultez Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Opérations de décalage de bits

Une opération de décalage de bits effectue un décalage arithmétique sur un modèle binaire. Ce modèle figure dans l'opérande de gauche, alors que l'opérande de droite indique le nombre de positions de décalage du modèle. Vous pouvez décaler le modèle vers la droite à l'aide de l'> >, opérateur (Visual Basic) ou vers la gauche à l'aide de l'< < opérateur (Visual Basic).

Le type de données de l'opérande de modèle doit être **SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**. Le type de données de l'opérande de décalage doit être **Integer** ou doit s'étendre à **Integer**.

Les décalages arithmétiques ne sont pas circulaires, ce qui signifie que les bits décalés à une extrémité du résultat ne sont pas réintroduits à l'autre extrémité. Les positions binaires libérées par un décalage sont définies comme suit :

- 0 pour un décalage arithmétique vers la gauche ;
- 0 pour le décalage arithmétique vers la droite d'un nombre positif ;
- 0 pour le décalage arithmétique vers la droite d'un type de données non signé (**Byte**, **UShort**, **UInteger**, **ULong**) ;
- 1 pour le décalage arithmétique vers la droite d'un nombre négatif (**SByte**, **Short**, **Integer** ou **Long**).

L'exemple suivant décale une valeur **Integer** vers la gauche et vers la droite.

VB

```
Dim lResult, rResult As Integer
Dim pattern As Integer = 12
' The low-order bits of pattern are 0000 1100.
lResult = pattern << 3
' A left shift of 3 bits produces a value of 96.
rResult = pattern >> 2
' A right shift of 2 bits produces value of 3.
```

Les décalages arithmétiques ne génèrent jamais d'exception de dépassement de capacité.

Opérateurs de bits

Les opérateurs logiques **Not**, **Or**, **And** et **Xor** effectuent également des opérations de bits arithmétiques lorsqu'ils sont utilisés sur des valeurs numériques. Pour plus d'informations, consultez « Opérations de bits » dans Opérateurs de bits et opérateurs logiques en Visual Basic.

Sécurité de type

Les opérandes doivent généralement être du même type. Par exemple, si vous ajoutez une variable **Integer**, vous devez l'ajouter à une autre variable **Integer**, et vous devez également assigner le résultat à une variable de type **Integer**.

Il est recommandé d'utiliser Option Strict, instruction pour assurer un codage correct de type sécurisé. Si vous définissez **Option Strict On**, Visual Basic exécute automatiquement des conversions *de type sécurisé*. Par exemple, si vous essayez d'ajouter une variable **Integer** à une variable **Double** et d'assigner la valeur à une variable **Double**, l'opération continue normalement, parce qu'une valeur **Integer** peut être convertie en **Double** sans perte de données. En revanche, les conversions de type non sécurisé provoquent une erreur de compilation avec **Option Strict On**. Par exemple, si vous tentez d'ajouter une variable **Integer** à une variable **Double** et d'assigner la valeur à une variable **Integer**, une erreur de compilation se produit, car une variable **Double** ne peut pas être implicitement convertie en type **Integer**.

Toutefois, si vous définissez **Option Strict Off**, Visual Basic autorise les conversions restrictives implicites, bien qu'elles puissent entraîner une perte inattendue de données ou de précision. C'est la raison pour laquelle nous vous recommandons d'utiliser **Option Strict On** lors de l'écriture du code de production. Pour plus d'informations, consultez Conversions étendues et restrictives.

Voir aussi

Référence

Opérateurs arithmétiques (Visual Basic)
Opérateurs de décalage de bits

Concepts

Opérateurs de comparaison en Visual Basic
Opérateurs de concaténation (Visual Basic)
Opérateurs de bits et opérateurs logiques en Visual Basic
Association efficace d'opérateurs

Opérateurs de comparaison en Visual Basic

Les opérateurs de comparaison comparent deux expressions et retournent une valeur **Boolean** qui représente la relation de leurs valeurs. Il existe des opérateurs pour comparer des valeurs numériques, des opérateurs pour comparer des chaînes et des opérateurs pour comparer des objets. Les trois types d'opérateurs sont détaillés ci-dessous.

Comparaison de valeurs numériques

Visual Basic compare des valeurs numériques à l'aide de six opérateurs de comparaison numériques. Chaque opérateur accepte comme opérandes deux expressions qui prennent des valeurs numériques. Le tableau suivant répertorie les opérateurs et affiche des exemples de chacun d'entre eux.

Opérateur	Condition testée	Exemples

= (égalité)	La valeur de la première expression est-elle égale à la valeur de la seconde ?	`23 = 33 ' False`
		`23 = 23 ' True`
		`23 = 12 ' False`

<> (inégalité)	La valeur de la première expression est-elle inégale à la valeur de la seconde ?	`23 <> 33 ' True`
		`23 <> 23 ' False`
		`23 <> 12 ' True`

< (inférieur à)	La valeur de la première expression est-elle inférieure à la valeur de la seconde ?	`23 < 33 ' True`
		`23 < 23 ' False`
		`23 < 12 ' False`

> (supérieur à)	La valeur de la première expression est-elle supérieure à la valeur de la seconde ?	`23 > 33 ' False`
		`23 > 23 ' False`
		`23 > 12 ' True`

<= (inférieur ou égal à)	La valeur de la première expression est-elle inférieure ou égale à la valeur de la seconde ?	`23 <= 33 ' True`
		`23 <= 23 ' True`
		`23 <= 12 ' False`

>= (supérieur ou égal à)	La valeur de la première expression est-elle supérieure ou égale à la valeur de la seconde ?	`23 >= 33 ' False`
		`23 >= 23 ' True`
		`23 >= 12 ' True`

Comparaison de chaînes

Visual Basic compare des chaînes à l'aide de l'opérateur Like et des opérateurs de comparaison numériques. L'opérateur **Like** vous permet de spécifier un modèle. La chaîne est ensuite comparée au modèle, et en cas de correspondance, le résultat est **True**. Sinon, le résultat est **False**. Les opérateurs numériques vous permettent de comparer des valeurs **String** en fonction de leur ordre de tri, comme illustré dans l'exemple ci-dessous :

```
"73" < "9"

' The result of the preceding comparison is True.
```

Le résultat de l'exemple précédent est **True** car le premier caractère de la première chaîne est trié avant le premier caractère de la deuxième chaîne. Si les premiers caractères sont identiques, la comparaison se poursuit jusqu'au prochain caractère des deux chaînes, etc. Vous pouvez également tester l'égalité des chaînes à l'aide de l'opérateur d'égalité, comme illustré dans l'exemple ci-dessous.

```
"734" = "734"

' The result of the preceding comparison is True.
```

Si une chaîne est un préfixe d'une autre chaîne (« aa » et « aaa », par exemple), la plus longue chaîne est considérée comme supérieure à la plus courte chaîne. L'exemple suivant illustre ce comportement :

```
"aaa" > "aa"

' The result of the preceding comparison is True.
```

L'ordre de tri dépend soit d'une comparaison binaire, soit d'une comparaison textuelle en fonction du paramètre **Option Compare**. Pour plus d'informations, consultez l'instruction Option Compare.

Comparaison d'objets

Visual Basic compare deux variables de référence d'objet à l'aide de l'opérateur Is (Visual Basic) et de l'opérateur IsNot. Vous pouvez utiliser l'un ou l'autre de ces opérateurs pour déterminer si deux variables de référence font référence à la même instance d'objet. L'exemple suivant illustre ce comportement :

VB

```
Dim x As testClass
Dim y As New testClass()
x = y
If x Is y Then
    ' Insert code to run if x and y point to the same instance.
End If
```

Dans l'exemple précédent, `x Is y` a la valeur **True**, parce que les deux variables font référence à la même instance. Comparez ce résultat à ce qui suit :

VB

```
Dim x As New customer()
Dim y As New customer()
If x Is y Then
    ' Insert code to run if x and y point to the same instance.
End If
```

Dans l'exemple précédent, `x Is y` a la valeur **False**, dans la mesure où, même si les variables font référence à des objets du même type, elles se rapportent à des instances distinctes de ce type.

Lorsque vous souhaitez tester deux objets qui ne pointent pas vers la même instance, l'opérateur **IsNot** vous permet d'éviter une combinaison grammaticalement maladroite de **Not** et **Is**. L'exemple suivant illustre ce comportement :

VB

```
Dim a As New classA()
Dim b As New classB()
If a IsNot b Then
    ' Insert code to run if a and b point to different instances.
End If
```

Dans l'exemple précédent, `If a IsNot b` est équivalent à `If Not a Is b`.

Comparaison du type d'objet

L'expression **TypeOf**...**Is** permet de tester si un objet est d'un type particulier. La syntaxe est la suivante :

```
TypeOf <objectexpression> Is <typename>
```

Lorsque `typename` spécifie un type interface, l'expression **TypeOf**...**Is** retourne **True** si l'objet implémente le type interface. Lorsque `typename` est un type classe, l'expression retourne **True** si l'objet est une instance de la classe spécifiée ou d'une classe qui dérive de la classe spécifiée. L'exemple suivant illustre ce comportement :

VB

```
Dim x As System.Windows.Forms.Button
x = New System.Windows.Forms.Button()
If TypeOf x Is System.Windows.Forms.Control Then
```

```
    ' Insert code to run if x is of type System.Windows.Forms.Control.
End If
```

Dans l'exemple précédent, l'expression `TypeOf x Is Control` a la valeur **True** car le type de `x` est `Button`, qui hérite de `Control`.

Pour plus d'informations, consultez TypeOf, opérateur (Visual Basic).

Voir aussi

Référence

Opérateurs de comparaison (Visual Basic)

Concepts

Comparaisons de valeurs

Opérateurs arithmétiques en Visual Basic

Opérateurs de concaténation (Visual Basic)

Opérateurs de bits et opérateurs logiques en Visual Basic

Autres ressources

Opérateurs (Visual Basic)

Comment : déterminer si deux objets sont identiques

Si vous avez deux variables qui font référence à des objets, vous pouvez utiliser l'opérateur **Is** ou **IsNot**, ou les deux, pour déterminer si elles font référence à la même instance.

Pour tester si deux objets sont identiques

- Utilisez l'opérateur Is (Visual Basic) ou l'opérateur IsNot avec les deux variables comme opérandes.

  VB

```
Public Sub processControl(ByVal f As System.Windows.Forms.Form, _
    ByVal c As System.Windows.Forms.Control)
    Dim active As System.Windows.Forms.Control = f.ActiveControl
    If (active IsNot Nothing) And (c Is active) Then
        ' Insert code to process control c
    End If
    Return
End Sub
```

Vous souhaitez éventuellement entreprendre une certaine action selon que deux objets font référence ou non à la même instance. L'exemple précédent compare le contrôle `c` au contrôle actif du formulaire `f`. S'il n'y a aucun contrôle actif, ou s'il y en a et qu'il ne s'agit pas de la même instance de contrôle que `c`, l'instruction **If** échoue et la procédure est retournée sans traitement supplémentaire.

L'utilisation de **Is** ou de **IsNot** est une question de commodité personnelle. L'un peut être plus être plus facile à lire que l'autre dans une expression donnée.

Voir aussi

Concepts

Opérateurs de comparaison en Visual Basic

Comment : faire correspondre une chaîne à un modèle

Pour savoir si une expression de String Data Type (Visual Basic) satisfait un modèle, vous pouvez utiliser l'opérateur Like.

Like accepte deux opérandes. L'opérande gauche est une expression de chaîne, et l'opérande droit est une chaîne contenant le modèle à utiliser pour la correspondance. **Like** retourne une valeur **Boolean** qui indique si l'expression de chaîne satisfait le modèle.

Vous pouvez faire correspondre chaque caractère de l'expression de chaîne avec un caractère spécifique, un caractère générique, une liste de caractères ou une plage de caractères. Les positions des spécifications dans la chaîne de masque correspondent aux positions des caractères de correspondance dans l'expression de chaîne.

Pour faire correspondre un caractère de l'expression de chaîne avec un caractère spécifique

- Mettez directement le caractère spécifique dans la chaîne de masque. Certains caractères spéciaux doivent être entourés par des crochets (**[ ]**). Pour plus d'informations, consultez Like, opérateur.

  L'exemple suivant teste si `myString` se compose exactement du caractère unique `H`.

  VB

```
Dim sMatch As Boolean = myString Like "H"
```

Pour faire correspondre un caractère de l'expression de chaîne avec un caractère générique

- Mettez un point d'interrogation (**?**) dans la chaîne de masque. Tout caractère valide dans cette position est mis en correspondance.

  L'exemple suivant teste si `myString` se compose du caractère unique `W`, suivi exactement de deux caractères d'une valeur quelconque.

  VB

```
Dim sMatch As Boolean = myString Like "W??"
```

Pour faire correspondre un caractère de l'expression de chaîne avec une liste de caractères

- Mettez des crochets (**[ ]**) dans la chaîne de masque, et insérez la liste de caractères à l'intérieur des crochets. Ne séparez pas les caractères par des virgules ou tout autre séparateur. Tout caractère unique de la liste est mis en correspondance.

  L'exemple suivant teste si `myString` se compose d'un caractère valide, suivi exactement de l'un des caractères `A`, `C` ou `E`.

  VB

```
Dim sMatch As Boolean = myString Like "?[ACE]"
```

  Notez que cette mise en correspondance respecte la casse.

Pour faire correspondre un caractère de l'expression de chaîne avec une plage de caractères

- Mettez des crochets (**[ ]**) dans la chaîne de masque, et à l'intérieur des crochets, mettez les caractères supérieur et inférieur de la plage, séparés par un trait d'union (**–**). Tout caractère unique de la plage est mis en correspondance.

  L'exemple suivant teste si `myString` se compose des caractères `num`, suivis exactement de l'un des caractères `i`, `j`, `k`, `l`, `m` ou `n`.

  VB

```
Dim sMatch As Boolean = myString Like "num[i-m]"
```

Notez que cette mise en correspondance respecte la casse.

Mise en correspondance de chaînes vides

Like traite la séquence `[]` comme une chaîne de longueur nulle (**""**). Vous pouvez utiliser `[]` pour tester si l'expression de chaîne entière est vide, mais vous ne pouvez pas l'utiliser pour tester si une position particulière de l'expression de chaîne est vide. Si une position vide est l'une des options que vous devez tester, vous pouvez utiliser **Like** plusieurs fois.

Pour faire correspondre un caractère de l'expression de chaîne avec une liste de caractères ou aucun caractère

1. Appelez deux fois l'opérateur **Like** sur la même expression de chaîne, et connectez les deux appels avec l'opérateur Or (Visual Basic) ou l'opérateur OrElse.
2. Dans la chaîne de masque de la première clause **Like**, incluez la liste de caractères, entourée par des crochets (**[ ]**).
3. Dans la chaîne de masque de la deuxième clause **Like**, ne mettez pas de caractères à la position concernée.

   L'exemple suivant teste le numéro de téléphone à sept chiffres `phoneNum` pour exactement trois chiffres, suivis d'un espace, d'un trait d'union (`-`), d'un point (`.`) ou d'aucun caractère, suivi d'exactement quatre chiffres.

   VB

```
Dim sMatch As Boolean = _
  (phoneNum Like "###[ -.]####") OrElse (phoneNum Like "#######")
```

Voir aussi

Référence

Opérateurs de comparaison (Visual Basic)
Like, opérateur
String, type de données (Visual Basic)

Concepts

Opérateurs et expressions en Visual Basic

Opérateurs de concaténation (Visual Basic)

Les opérateurs de concaténation relient plusieurs chaînes dans une seule chaîne. Il existe deux opérateurs de concaténation, **+** et **&**. Les deux exécutent l'opération de concaténation de base, comme le montre l'exemple suivant.

VB

```
Dim x As String = "Con" & "caten" & "ation"
Dim y As String = "Con" + "caten" + "ation"
' The preceding statements set both x and y to "Concatenation".
```

Ces opérateurs peuvent également concaténer des variables **String**, comme le montre l'exemple suivant :

VB

```
Dim a As String = "abc"
Dim d As String = "def"
Dim z As String = a & d
Dim w As String = a + d
' The preceding statements set both z and w to "abcdef".
```

Différences entre les deux opérateurs de concaténation

L'+, opérateur (Visual Basic) a pour objectif principal d'ajouter deux nombres. Toutefois, il peut également concaténer des opérandes numériques avec des opérandes de chaîne. L'opérateur **+** a un jeu de règles complexes qui déterminent s'il faut ajouter, concaténer, signaler une erreur du compilateur ou lever une exception InvalidCastException au moment de l'exécution.

L'opérateur &, opérateur (Visual Basic) n'est défini que pour des opérandes **String** et étend toujours l'opérande associé au type **String**, quelle que soit la valeur de **Option Strict**. L'opérateur **&** est recommandé pour la concaténation de chaînes, car il est défini exclusivement pour les chaînes et il réduit le risque de générer une conversion inattendue.

Performances : chaîne et StringBuilder

Si vous effectuer un nombre considérable de manipulations sur une chaîne, telles que des concaténations, des suppressions et des remplacements, vos performances peuvent tirer profit de la classe StringBuilder de l'espace de noms System.Text. Elle prend une instruction supplémentaire pour créer et initialiser un objet **StringBuilder** et une autre instruction pour convertir sa dernière valeur en un **String**, mais vous pouvez effectuer une récupération parce que **StringBuilder** peut s'exécuter plus rapidement.

Voir aussi

Référence

Option Strict, instruction

Concepts

Types de méthodes de manipulation de chaînes en Visual Basic
Opérateurs arithmétiques en Visual Basic
Opérateurs de comparaison en Visual Basic
Opérateurs de bits et opérateurs logiques en Visual Basic

Opérateurs de bits et opérateurs logiques en Visual Basic

Les opérateurs logiques comparent des expressions **Boolean** et retournent un résultat **Boolean**. Les opérateurs **And**, **Or**, **AndAlso**, **OrElse** et **Xor** sont *binaires* parce qu'ils prennent deux opérandes, tandis que l'opérateur **Not** est *unaire* parce qu'il prend une seule opérande. Certains de ces opérateurs peuvent également exécuter des opérations logiques de bits sur des valeurs intégrales.

Opérateur logique unaire

L'Not, opérateur (Visual Basic) effectue une *négation* logique sur une expression **Boolean**. Il retourne l'opposé logique de son opérande. Si l'expression a la valeur **True**, **Not** retourne **False** ; si elle a la valeur **False**, **Not** retourne **True**. L'exemple suivant illustre ce comportement :

VB

```
Dim x, y As Boolean
x = Not 23 > 14
y = Not 23 > 67
' The preceding statements set x to False and y to True.
```

Opérateurs logiques binaires

L'And, opérateur (Visual Basic) effectue une *conjonction* logique sur deux expressions **Boolean**. Si les deux expressions ont la valeur **True**, **And** retourne **True**. Si au moins l'une des expressions a la valeur **False**, **And** retourne **False**.

L'Or, opérateur (Visual Basic) effectue une *disjonction* ou une *inclusion* logique sur deux expressions **Boolean**. Si l'une des deux expressions a la valeur **True**, ou encore si elles ont toutes les deux la valeur **True**, **Or** retourne **True**. Si aucune expression n'a la valeur **True**, **Or** retourne **False**.

L'Xor, opérateur (Visual Basic) effectue une *exclusion* logique sur deux expressions **Boolean**. Si une seule expression a la valeur **True**, **Xor** retourne **True**. Si les deux expressions ont la valeur **True** ou la valeur **False**, **Xor** retourne **False**.

L'exemple suivant illustre les opérateurs **And**, **Or** et **Xor**.

VB

```
Dim a, b, c, d, e, f, g As Boolean

a = 23 > 14 And 11 > 8
b = 14 > 23 And 11 > 8
' The preceding statements set a to True and b to False.

c = 23 > 14 Or 8 > 11
d = 23 > 67 Or 8 > 11
' The preceding statements set c to True and d to False.

e = 23 > 67 Xor 11 > 8
f = 23 > 14 Xor 11 > 8
g = 14 > 23 Xor 8 > 11
' The preceding statements set e to True, f to False, and g to False.
```

Opérations logiques de court-circuit

L'AndAlso, opérateur est très similaire à l'opérateur **And**, car il effectue également une conjonction logique sur deux expressions **Boolean**. La différence principale entre les deux est que **AndAlso** se comporte comme un *court-circuit*. Si la première expression d'une expression **AndAlso** a la valeur **False**, la deuxième expression n'est pas évaluée, car elle ne peut pas modifier le résultat final et **AndAlso** retourne **False**.

De même, l'OrElse, opérateur effectue une disjonction logique de court-circuit sur deux expressions **Boolean**. Si la première expression d'une expression **OrElse** a la valeur **True**, la deuxième expression n'est pas évaluée, car elle ne peut pas modifier le résultat final et **OrElse** retourne **True**.

Compromis de court-circuit

Un court-circuit peut améliorer les performances en n'évaluant pas une expression qui ne peut pas modifier le résultat de l'opération logique. Toutefois, si cette expression exécute des actions supplémentaires, le court-circuit les ignore. Par exemple, si l'expression inclut un appel à une procédure **Function**, cette procédure n'est pas appelée si l'expression est court-circuitée, et le code supplémentaire contenu dans la **Function** ne s'exécute pas. Si votre logique de programme dépend de l'un de ces codes supplémentaires, vous devez probablement éviter de court-circuiter les opérateurs.

L'exemple suivant illustre la différence entre **And**, **Or** et leurs équivalents de court-circuit.

VB

```
Dim amount As Integer = 12
Dim highestAllowed As Integer = 45
Dim grandTotal As Integer
```

VB

```
If amount > highestAllowed And checkIfValid(amount) Then
    ' The preceding statement calls checkIfValid().
End If
If amount > highestAllowed AndAlso checkIfValid(amount) Then
    ' The preceding statement does not call checkIfValid().
End If
If amount < highestAllowed Or checkIfValid(amount) Then
    ' The preceding statement calls checkIfValid().
End If
If amount < highestAllowed OrElse checkIfValid(amount) Then
    ' The preceding statement does not call checkIfValid().
End If
```

VB

```
Function checkIfValid(ByVal checkValue As Integer) As Boolean
    If checkValue > 15 Then
        MsgBox(CStr(checkValue) & " is not a valid value.")
        ' The MsgBox warning is not displayed if the call to
        ' checkIfValid() is part of a short-circuited expression.
        Return False
    Else
        grandTotal += checkValue
        ' The grandTotal value is not updated if the call to
        ' checkIfValid() is part of a short-circuited expression.
        Return True
    End If
End Function
```

Dans l'exemple précédent, notez que du code important dans `checkIfValid()` ne s'exécute pas lorsque l'appel est court-circuité. La première instruction **If** appelle `checkIfValid()` bien que `12 > 45` retourne **False**, parce que **And** n'effectue pas de court-circuit. La deuxième instruction **If** n'appelle pas `checkIfValid()` parce que si `12 > 45` retourne **False**, **AndAlso** court-circuite la deuxième expression. La troisième instruction **If** appelle `checkIfValid()` bien que `12 < 45` retourne **True**, parce que **Or** n'effectue pas de court-circuit. La quatrième instruction **If** n'appelle pas `checkIfValid()` parce que si `12 < 45` retourne **True**, **OrElse** court-circuite la deuxième expression.

Opérateurs de bits

Les opérations de bits évaluent deux valeurs intégrales sous forme binaire (base 2). Elles comparent les bits à des positions correspondantes, puis assignent des valeurs en fonction de la comparaison. L'exemple suivant illustre l'opérateur **And**.

```
Dim x As Integer
x = 3 And 5
```

L'exemple précédent affecte à `x` la valeur 1. Cela est dû aux raisons suivantes :

- Les valeurs sont traitées comme des valeurs binaires :

3 sous forme binaire = 011

5 sous forme binaire = 101

- L'opérateur **And** compare les représentations binaires par position binaire (bit). Si les deux bits d'une position donnée correspondent à 1, une valeur égale à 1 est placée à cette position dans le résultat. Si l'un des deux bits est 0, une valeur égale à 0 est placée à cette position dans le résultat. Dans l'exemple précédent, cette procédure fonctionne comme suit :

  011 (3 sous forme binaire)

  101 (5 sous forme binaire)

  001 (le résultat sous forme binaire)

- Le résultat est traité comme une valeur décimale. La valeur 001 est la représentation binaire de 1, donc `x` = 1.

L'opération **Or** au niveau des bits est similaire, excepté que 1 est assigné au bit de résultat si l'un des deux bits comparés est égal à 1. **Xor** assigne 1 au bit de résultat si un seul des bits comparés est égal à 1. **Not** accepte un opérande simple, inverse tous les bits (y compris le bit de signe) et assigne la valeur au résultat. Ainsi, pour les nombres positifs signés, **Not** retourne toujours une valeur négative, et pour les nombres négatifs, **Not** retourne toujours une valeur positive ou nulle.

Les opérateurs **AndAlso** et **OrElse** ne prennent pas en charge les opérations de bits.

Remarque
Les opérations de bits ne peuvent être effectuées que sur des types intégraux. Les valeurs de virgule flottante doivent être co nverties en types intégraux avant de poursuivre une opération de bits.

Voir aussi

Référence

Opérateurs logiques/de bits

Concepts

Expressions booléennes

Opérateurs arithmétiques en Visual Basic

Opérateurs de comparaison en Visual Basic

Opérateurs de concaténation (Visual Basic)

Association efficace d'opérateurs

Association efficace d'opérateurs

Des expressions complexes peuvent contenir plusieurs opérateurs distincts. L'exemple suivant illustre ce comportement :

```
x = (45 * (y + z)) ^ (2 / 85) * 5 + z
```

La création d'expressions complexes comme celle de l'exemple précédent nécessite de bien connaître les règles de priorité des opérateurs. Pour plus d'informations, consultez Priorité des opérateurs en Visual Basic.

Expressions entre parenthèses

Vous voulez parfois que les opérations soient effectuées dans un ordre différent de celui déterminé par la priorité des opérateurs. Prenons l'exemple suivant.

```
x = z * y + 4
```

L'exemple précédent multiplie `z` par `y`, puis ajoute le résultat à `4`. Mais si vous souhaitez ajouter `y` et `4` avant de multiplier le résultat par `z`, vous pouvez substituer la priorité des opérateurs normale en utilisant des parenthèses. En effet, si vous insérez des parenthèses autour d'une expression, cette expression sera d'abord évaluée, sans tenir compte de la priorité des opérateurs. Pour forcer l'exemple précédent à effectuer l'addition en premier lieu, vous pouvez le réécrire comme dans l'exemple suivant.

```
x = z * (y + 4)
```

L'exemple précédent ajoute `y` et `4`, puis multiplie cette somme par `z`.

Expressions entre parenthèses imbriquées

Vous pouvez imbriquer les expressions dans plusieurs niveaux de parenthèses pour substituer davantage la priorité. Les expressions les plus imbriquées dans les parenthèses sont évaluées en premier lieu, suivies des expressions suivantes les plus imbriquées, etc. jusqu'à celles les moins imbriquées et, enfin, des expressions à l'extérieur des parenthèses. L'exemple suivant illustre ce comportement :

```
x = (z * 4) ^ (y * (z + 2))
```

Dans l'exemple précédent, `z + 2` est évalué en premier lieu, suivi des autres expressions entre parenthèses. L'élévation à la puissance, qui est en principe prioritaire par rapport à l'addition ou à la multiplication, est évaluée en dernier lieu dans cet exemple, car les autres expressions figurent entre parenthèses.

Voir aussi

Tâches

Comment : calculer des valeurs numériques

Référence

Opérateurs logiques/de bits

Priorité des opérateurs en Visual Basic

Concepts

Opérateurs arithmétiques en Visual Basic

Opérateurs de comparaison en Visual Basic

Opérateurs de bits et opérateurs logiques en Visual Basic

Expressions booléennes

Comparaisons de valeurs

Comment : calculer des valeurs numériques

Vous pouvez calculer des valeurs numériques à l'aide d'expressions numériques. Une *expression numérique* contient des littéraux, des constantes et des variables représentant des valeurs numériques, ainsi que des opérateurs qui agissent sur ces valeurs.

Calcul de valeurs numériques

Pour calculer une valeur numérique

- Combinez un ou plusieurs littéraux, constantes et variables numériques dans une expression numérique. Vous trouverez ci-dessous des exemples d'expressions numériques valides.

  ```
  93.217

  System.Math.PI

  counter

  4 * (67 + i)
  ```

  Les trois premières lignes affichent un littéral, une constante et une variable. Chacune d'elle forme une expression numérique valide par elle-même. La dernière ligne affiche une combinaison d'une variable avec deux littéraux.

  Notez qu'une expression numérique ne forme pas d'instruction Visual Basic complète par elle-même. Vous devez utiliser l'expression dans le cadre d'une instruction complète.

Pour stocker une valeur numérique

- Vous pouvez utiliser une instruction d'assignation pour assigner la valeur représentée par une expression numérique à une autre variable, comme illustré dans cet exemple.

  VB

  ```
  Dim i As Integer = 2
  Dim j As Integer
  j = 4 * (67 + i)
  ```

  Dans l'exemple précédent, la valeur de l'expression située à droite de l'opérateur égal (**=**) (equals) est assignée à la variable `j` située à gauche de l'opérateur ; par conséquent, `j` a la valeur 276.

  Pour plus d'informations, consultez Instructions d'assignation.

Opérateurs multiples

Si l'expression numérique contient plusieurs opérateurs, ceux-ci sont évalués en fonction de l'ordre déterminé par les règles de priorité des opérateurs. Pour substituer ces règles de priorité d'opérateurs, vous devez insérer des parenthèses autour des expressions, comme le montre l'exemple précédent ; les expressions insérées sont évaluées en premier lieu.

Pour substituer la priorité d'opérateurs normale

- Utilisez des parenthèses pour insérer les opérations à effectuer en premier lieu. L'exemple suivant affiche deux résultats différents avec les mêmes opérandes et opérateurs.

  VB

  ```
  Dim i As Integer = 2
  Dim j, k As Integer
  j = 4 * (67 + i)
  k = 4 * 67 + i
  ```

Dans l'exemple précédent, le calcul de `j` exécute d'abord l'opérateur d'addition (**+**) parce que les parenthèses qui entourent `(67 + i)` substituent la priorité normale et la valeur assignée à `j` est de 276 (4 fois 69). Le calcul de `k` exécute les opérateurs dans leur priorité normale (***** avant **+**) et la valeur assignée à `k` est de 270 (268 plus 2).

Pour plus d'informations, consultez Priorité des opérateurs en Visual Basic.

Voir aussi

Référence

Priorité des opérateurs en Visual Basic
Opérateurs arithmétiques (Visual Basic)

Concepts

Opérateurs et expressions en Visual Basic
Comparaisons de valeurs
Instructions d'assignation
Association efficace d'opérateurs

Comparaisons de valeurs

Vous pouvez utiliser des opérateurs de comparaison pour construire des expressions qui comparent les valeurs de variables numériques. Ces expressions retournent une valeur **Boolean** selon que la comparaison est vraie (true) ou fausse (false). Des exemples d'une telle expression sont les suivants :

```
45 > 26
```

```
26 > 45
```

La première expression prend la valeur **True**, car 45 est supérieur à 26. Le second exemple prend la valeur **False**, car 26 est inférieur à 45.

Vous pouvez également comparer des expressions numériques de cette façon. Les expressions comparées peuvent être des expressions complexes, comme dans l'exemple suivant :

```
x / 45 * (y +17) >= System.Math.Sqrt(z) / (p - (x * 16))
```

La précédente expression complexe inclut des littéraux, des variables et des appels de fonction. Les expressions sont évaluées des deux côtés de l'opérateur de comparaison, et les résultats sont ensuite comparés par l'opérateur de comparaison **>=**. Si la valeur de l'expression de gauche est supérieure ou égale à la valeur de l'expression de droite, l'expression complète a la valeur **True** ; sinon, elle correspond à **False**.

Les expressions qui comparent des valeurs sont plus généralement utilisées dans des constructions **If...Then**, comme dans l'exemple suivant :

VB

```
If x > 50 Then
    ' Insert code to run if x is greater than 50.
Else
    ' Insert code to run if x is less than or equal to 50.
End If
```

Le signe **=** est un opérateur de comparaison et un opérateur d'assignation. Lorsque ce signe est utilisé comme un opérateur de comparaison, il évalue si la valeur de gauche est égale à la valeur de droite, comme le montre l'exemple suivant :

VB

```
If x = 50 Then
    ' Insert code to continue program.
End If
```

Vous pouvez également utiliser une expression de comparaison lorsqu'une valeur **Boolean** est requise (dans une instruction **If**, **While**, **Loop** ou **ElseIf**, par exemple) ou lors de l'assignation ou du passage d'une valeur à une variable **Boolean**. Dans l'exemple suivant, la valeur retournée par l'expression de comparaison est assignée à une variable **Boolean** :

VB

```
Dim x As Boolean
x = 50 < 30
' The preceding statement assigns False to x.
```

Voir aussi

Tâches

Comment : calculer des valeurs numériques

Référence

Priorité des opérateurs en Visual Basic

Concepts

Expressions booléennes

Opérateurs et expressions en Visual Basic

Opérateurs de comparaison en Visual Basic

Expressions booléennes

Une *expression booléenne* est une expression qui a pour valeur le type de données booléen. Les expressions **Boolean** peuvent revêtir plusieurs formes. La forme la plus simple est la comparaison directe de la valeur d'une variable **Boolean** à un littéral **Boolean**, comme le montre l'exemple suivant :

VB

```
If newCustomer = True Then
    ' Insert code to execute if newCustomer = True.
Else
    ' Insert code to execute if newCustomer = False.
End If
```

Deux significations de l'opérateur =

Notez que l'instruction d'assignation `newCustomer = True` ressemble à l'expression de l'exemple précédent, mais elle effectue une fonction différente et est utilisée différemment. Dans l'exemple précédent, l'expression `newCustomer = True` représente une valeur Boolean et le signe **=** est interprété comme un opérateur de comparaison. Dans une instruction autonome, le signe **=** est interprété comme un opérateur d'assignation et assigne la valeur de droite à la variable de gauche. L'exemple suivant illustre ce comportement :

VB

```
If newCustomer = True Then
    newCustomer = False
End If
```

Pour plus d'informations, consultez Comparaisons de valeurs et Instructions d'assignation.

Opérateurs de comparaison

Les opérateurs de comparaison (**=**, **<**, **>**, **<>**, **<=** et **>=**) produisent des expressions booléennes en comparant l'expression située à gauche de l'opérateur à l'expression de droite et en assignant la valeur **True** ou **False** au résultat. L'exemple suivant illustre ce comportement :

```
42 < 81
```

Étant donné que 42 est inférieur à 81, l'expression booléenne de l'exemple précédent a la valeur **True**. Pour plus d'informations sur ce type d'expression, consultez Comparaisons de valeurs.

Opérateurs de comparaison associés aux opérateurs logiques

Les expressions de comparaison peuvent être combinées à l'aide d'opérateurs logiques pour créer des expressions booléennes plus complexes. L'exemple suivant illustre l'utilisation d'opérateurs de comparaison associés à un opérateur logique.

```
x > y And x < 1000
```

Dans cet exemple, la valeur de l'expression globale dépend des valeurs des expressions de chaque côté de l'opérateur **And**. Si les deux expressions ont la valeur **True**, l'expression globale a la valeur **True**. Si l'une des expressions a la valeur **False**, l'expression globale a la valeur **False**.

Opérateurs de court-circuit

Les opérateurs logiques **AndAlso** et **OrElse** se comportent comme des *courts-circuits*. Un opérateur de court-circuit évalue premièrement l'opérande de gauche. Si l'opérande de gauche détermine la valeur de l'expression globale, l'exécution du programme se poursuit sans évaluer l'expression de droite. L'exemple suivant illustre ce comportement :

VB

```
If 45 < 12 AndAlso testFunction(3) = 81 Then
    ' Add code to continue execution.
End If
```

Dans l'exemple précédent, l'opérateur évalue l'expression de gauche, `45 < 12`. Puisque l'expression de gauche a pour valeur **False**, l'expression logique globale doit avoir la valeur **False**. L'exécution du programme ignore ainsi l'exécution du code dans le bloc **If** sans évaluer l'expression de droite, `testFunction(3)`. Cet exemple n'appelle pas `testFunction()` parce que l'expression de gauche falsifie l'expression globale.

De même, si l'expression de gauche d'une expression logique utilisant **OrElse** a la valeur **True**, l'exécution se poursuit sur la prochaine ligne de code sans évaluer l'expression de droite, car l'expression de gauche a déjà validé l'expression complète.

Comparaison avec des opérateurs n'effectuant pas de court-circuit

En revanche, les deux côtés de l'opérateur logique sont évalués lors de l'utilisation des opérateurs logiques **And** et **Or**. L'exemple suivant illustre ce comportement :

VB

```
If 45 < 12 And testFunction(3) = 81 Then
    ' Add code to continue execution.
End If
```

L'exemple précédent appelle `testFunction()` même si l'expression de gauche a la valeur **False**.

Expressions entre parenthèses

Vous pouvez utiliser des parenthèses pour contrôler l'ordre d'évaluation des expressions booléennes. Les expressions entre parenthèses sont évaluées en premier lieu. Pour plusieurs niveaux d'imbrication, la priorité est accordée aux expressions les plus profondément imbriquées. Au sein des parenthèses, l'évaluation se poursuit selon les règles de priorité des opérateurs. Pour plus d'informations, consultez Priorité des opérateurs en Visual Basic.

Voir aussi

Référence

Opérateurs de comparaison (Visual Basic)
Priorité des opérateurs en Visual Basic
Boolean, type de données (Visual Basic)

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic
Comparaisons de valeurs
Instructions d'assignation
Association efficace d'opérateurs

Instructions dans Visual Basic

En Visual Basic, une *instruction* est une instruction complète pouvant contenir des mots clés, des opérateurs, des variables, des constantes et des expressions. Toutes les instructions sont classées en deux catégories : les *instructions de déclaration* (qui nomment une variable, une constante ou une procédure et qui peuvent également spécifier un type de données) ou les *instructions exécutables* (qui initient des actions).

Dans cette section

Vue d'ensemble des instructions

Offre une vue d'ensemble de haut niveau sur les instructions, y compris la façon d'inclure plusieurs instructions sur une ligne, de continuer une instruction sur plusieurs lignes, d'ajouter des commentaires et de vérifier les erreurs de compilation.

Instructions d'assignation

Décrit les instructions qui réalisent des opérations d'assignation, qui assignent la valeur située à droite de l'opérateur à celle de gauche.

Instructions de déclaration en Visual Basic

Décrit les instructions permettant de nommer et de définir des procédures, des variables, des tableaux et des constantes.

Instructions exécutables

Décrit les instructions qui lancent une action en exécutant une méthode. Les instructions exécutables peuvent parcourir ou créer des branches de blocs de code et contiennent souvent des opérateurs mathématiques ou conditionnels.

Rubriques connexes

Opérateurs d'assignation

Fournit des liens vers des pages de référence du langage qui décrivent les opérateurs d'assignation, tels que **=**, ***=** et **&=**.

Opérateurs et expressions en Visual Basic

Explique comment combiner des éléments avec des opérateurs pour retourner de nouvelles valeurs.

Vue d'ensemble des instructions

En Visual Basic, une instruction est une instruction complète. Elle contient des mots clés, des opérateurs, des variables, des constantes et des expressions. Chaque instruction appartient à l'une des catégories suivantes :

- instructions de déclaration qui nomment une variable, une constante ou une procédure et qui peuvent également spécifier un type de données ;
- instructions exécutables qui initient des actions. Ces instructions peuvent appeler une méthode ou une fonction, et peuvent parcourir ou créer des branches de blocs de code. Les instructions exécutables contiennent des instructions d'assignation, qui assignent une valeur ou une expression à une variable ou à une constante.

Placement de plusieurs instructions sur une seule ligne

Plusieurs instructions peuvent figurer sur une seule ligne, séparées par le signe deux-points (**:**). L'exemple suivant illustre ce comportement.

VB

```
Dim sampleString As String = "Hello World" : MsgBox(sampleString)
```

Ce type de syntaxe rend souvent la lecture et la gestion du code difficiles. Il est donc recommandé de conserver une instruction sur une ligne.

Suite d'une instruction sur plusieurs lignes

Une instruction figure généralement dans une seule ligne, mais lorsqu'elle est trop longue, vous pouvez la poursuivre sur la ligne suivante en utilisant une séquence de continuation de ligne, composée respectivement d'un espace, d'un trait de soulignement (_) et d'un retour chariot. Dans l'exemple suivant, l'instruction exécutable **MsgBox** se poursuit sur deux lignes.

VB

```
Public Sub demoBox()
    Dim nameVar As String
    nameVar = "John"
    MsgBox("Hello " & nameVar & _
        ". How are you?")
End Sub
```

Ajout de commentaires

Le code source n'est pas toujours explicatif, même pour le programmeur qui l'a écrit. Pour simplifier la documentation de leur code, la majorité des programmeurs utilisent donc de nombreux commentaires incorporés. Les commentaires du code peuvent expliquer une procédure ou une instruction particulière à n'importe quelle personne qui la lit ou l'utilise ultérieurement. Visual Basic ignore les commentaires pendant la compilation et ils n'affectent pas le code compilé.

Les lignes des commentaires commencent par une apostrophe (') ou par **REM** suivi d'un espace. Ils peuvent être ajoutés n'importe où dans le code, sauf dans une chaîne. Pour ajouter un commentaire à une instruction, insérez une apostrophe ou **REM** après l'instruction, suivie du commentaire. Les commentaires peuvent également figurer sur leur propre ligne distincte. L'exemple de code suivant illustre ces possibilités.

VB

```
' This is a comment on a separate code line.
REM This is another comment on a separate code line.
x += a(i) * b(i) ' Add this amount to total.
MsgBox(statusMessage) REM Inform operator of status.
```

Vérification des erreurs de compilation

Si après avoir tapé une ligne de code, la ligne est affichée avec un soulignement ondulant (un message d'erreur peut

également être affiché), l'instruction comporte une erreur de syntaxe. Vous devez localiser le problème de l'instruction (en recherchant dans la liste des tâches ou en plaçant le pointeur de la souris sur l'erreur pour lire le message d'erreur) et le corriger. Votre programme ne sera pas correctement compilé si vous ne corrigez pas toutes les erreurs de syntaxe dans votre code.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code
Comment : étiqueter des instructions

Concepts

Instructions d'assignation
Instructions de déclaration en Visual Basic
Instructions exécutables

Instructions d'assignation

Les instructions d'assignation exécutent des opérations d'assignation qui consistent à prendre la valeur située à droite de l'opérateur d'assignation (**=**) et à la stocker dans l'élément situé à gauche, comme dans l'exemple suivant.

VB

```
v = 42
```

Dans l'exemple précédent, l'instruction d'assignation stocke la valeur littérale 42 dans la variable `v`.

Éléments de programmation admissibles

L'élément de programmation situé à gauche de l'opérateur d'assignation doit être en mesure d'accepter et de stocker une valeur. En d'autres termes, il doit être une variable ou une propriété qui n'est pas ReadOnly (Visual Basic), ou encore il doit être un élément de tableau. Dans le contexte d'une instruction d'assignation, un tel élément est parfois appelé *lvalue*, pour « valeur de gauche ».

La valeur située à droite de l'opérateur d'assignation est générée par une expression qui peut être composée de n'importe quelle combinaison de littéraux, de constantes, de variables, de propriétés, d'éléments de tableau, d'autres expressions ou d'appels de fonctions. L'exemple suivant illustre ce comportement.

VB

```
x = y + z + findResult(3)
```

L'exemple précédent ajoute la valeur contenue dans la variable `y` à celle qui est contenue dans la variable `z`variable, puis ajoute la valeur retournée par l'appel à la fonction `findResult`. La valeur totale de cette expression est ensuite stockée dans la variable `x`.

Types de données dans les instructions d'assignation

Outre les valeurs numériques, l'opérateur d'assignation peut également assigner des valeurs **String**, comme le montre l'exemple suivant.

VB

```
Dim a, b As String
a = "String variable assignment"
b = "Con" & "cat" & "enation"
' The preceding statement assigns the value "Concatenation" to b.
```

Vous pouvez également assigner des valeurs **Boolean** à l'aide d'un littéral **Boolean** ou d'une expression **Boolean**, comme le montre l'exemple suivant.

VB

```
Dim r, s, t As Boolean
r = True
s = 45 > 1003
t = 45 > 1003 Or 45 > 17
' The preceding statements assign False to s and True to t.
```

De même, vous pouvez assigner des valeurs appropriées aux éléments de programmation du type de données **Char**, **Date** ou **Object**. Vous pouvez également assigner une instance d'objet à un élément a déclaré pour faire partie de la classe dans laquelle cette instance est créée.

Instructions d'assignation composée

Les *instructions d'assignation composée* exécutent en premier lieu une opération sur une expression avant de l'assigner à un

élément de programmation. L'exemple suivant illustre l'un de ces opérateurs, **+=**, qui incrémente la valeur de la variable située à gauche de l'opérateur par la valeur de l'expression de droite.

VB

```
n += 1
```

L'exemple précédent ajoute 1 à la valeur de `n`, puis stocke cette nouvelle valeur dans `n`. C'est l'équivalent abrégé de l'instruction suivante :

VB

```
n = n + 1
```

Les opérateurs de ce type permettent d'effectuer de nombreuses opérations d'assignation composées. Pour obtenir la liste de ces opérateurs et en savoir davantage sur eux, consultez Opérateurs d'assignation.

Enfin, l'opérateur d'assignation de concaténation (**&=**) est utile pour ajouter une chaîne à la fin de chaînes déjà existantes, comme le montre l'exemple suivant :

VB

```
Dim q As String = "Sample "
q &= "String"
' q now contains "Sample String".
```

Conversions de types dans les instructions d'assignation

La valeur que vous assignez à une variable, une propriété ou un élément de tableau doit être un type de données approprié pour cet élément de destination. En général, vous devez essayer de générer une valeur du même type de données que celui de l'élément de destination. Toutefois, certains types peuvent être convertis en d'autres types au cours de l'assignation.

Pour plus d'informations sur la conversion entre types de données, consultez Conversions de type en Visual Basic. En bref, Visual Basic convertit automatiquement une valeur d'un type donné en un autre type auquel elle s'étend. Une *conversion étendue* réussit toujours au moment de l'exécution et ne perd pas de données. Par exemple, Visual Basic convertit une valeur **Integer** en **Double** lorsque cela s'avère nécessaire, parce que **Integer** s'étend à **Double**. Pour plus d'informations, consultez Conversions étendues et restrictives.

Les *conversions restrictives* (celles qui ne sont pas étendues) courent un risque d'échec au moment de l'exécution, ou de perte de données. Vous pouvez effectuer explicitement une conversion restrictive en utilisant une fonction de conversion de type, ou encore demander au compilateur d'effectuer implicitement toutes les conversions en définissant **Option Strict Off**. Pour plus d'informations, consultez Conversions implicites et explicites.

Voir aussi

Référence

Opérateurs d'assignation

Concepts

Vue d'ensemble des instructions
Instructions de déclaration en Visual Basic
Instructions exécutables
Conversions étendues et restrictives

Instructions de déclaration en Visual Basic

Les instructions de déclaration permettent de nommer et de définir des procédures, des variables, des propriétés, des tableaux et des constantes. Lorsque vous déclarez un élément de programmation, vous pouvez également définir son type de données, son niveau d'accès et sa portée. Pour plus d'informations, consultez Caractéristiques d'éléments déclarés.

L'exemple suivant contient trois déclarations.

VB

```
Public Sub applyFormat()
    Const limit As Integer = 33
    Dim thisWidget As New widget
    ' Insert code to implement the procedure.
End Sub
```

La première déclaration est l'instruction **Sub**. Associée à son instruction **End Sub** correspondante, elle déclare une procédure nommée `applyFormat`. Elle spécifie également que `applyFormat` a la valeur **Public**, ce qui signifie qu'un code qui peut y faire référence peut l'appeler.

La deuxième déclaration est l'instruction **Const** qui déclare la constante `limit`, en spécifiant le type de données **Integer** et la valeur 33.

La troisième déclaration est l'instruction **Dim**, qui déclare la variable `thisWidget`. Le type de données est un objet spécifique, à savoir un objet créé à partir de la classe `Widget`. Vous pouvez déclarer une variable comme étant d'un type de données élémentaire ou d'un type d'objet exposé dans l'application utilisée.

Valeurs initiales

Lorsque le code contenant une instruction de déclaration s'exécute, Visual Basic réserve la mémoire requise pour l'élément déclaré. Si l'élément contient une valeur, Visual Basic l'initialise sur la valeur par défaut de son type de données. Pour plus d'informations, consultez « Comportement » dans Dim, instruction (Visual Basic).

Vous pouvez assigner une valeur initiale à une variable dans le cadre de sa déclaration, comme l'exemple suivant l'illustre.

VB

```
Dim m As Integer = 45
' The preceding declaration creates m and assigns the value 45 to it.
```

Si votre variable est une variable objet, vous pouvez explicitement créer une instance de sa classe lorsque vous la déclarez à l'aide du mot clé New (Visual Basic), comme indiqué dans l'exemple suivant

VB

```
Dim f As New System.Windows.Forms.Form()
```

Notez que la valeur initiale que vous spécifiez dans une instruction de déclaration n'est pas assignée à une variable tant que l'exécution n'atteint pas son instruction de déclaration. Jusqu'à ce moment, la variable contient la valeur par défaut de son type de données.

Voir aussi

Tâches

Comment : déclarer une constante

Concepts

Vue d'ensemble des instructions

Instructions d'assignation

Instructions exécutables

Instructions exécutables

Une instruction exécutable effectue une action. Elle peut appeler une procédure, effectuer un branchement vers un autre emplacement du code, parcourir plusieurs instructions ou évaluer une expression. Une instruction d'assignation est un cas spécial d'une instruction exécutable.

L'exemple suivant utilise une structure de contrôle **If...Then...Else** pour exécuter divers blocs de code selon la valeur d'une variable. Une boucle **For...Next** s'exécute un certain nombre de fois dans chaque bloc de code.

VB

```
Public Sub startWidget(ByVal aWidget As widget, _
    ByVal clockwise As Boolean, ByVal revolutions As Integer)
    Dim counter As Integer
    If clockwise = True Then
        For counter = 1 To revolutions
            aWidget.spinClockwise()
        Next counter
    Else
        For counter = 1 To revolutions
            aWidget.spinCounterClockwise()
        Next counter
    End If
End Sub
```

L'instruction **If**de l'exemple précédent vérifie la valeur du paramètre `clockwise`. Si la valeur est **True**, elle appelle la méthode `spinClockwise` de `aWidget`. Si la valeur est **False**, elle appelle la méthode `spinCounterClockwise` de `aWidget`. La structure de contrôle **If...Then...Else** se termine par **End If**.

La boucle **For...Next** à l'intérieur de chaque bloc appelle la méthode appropriée autant de fois que la valeur du paramètre `revolutions`.

Voir aussi

Concepts

Vue d'ensemble des instructions
Instructions d'assignation
Instructions de déclaration en Visual Basic

Procédures dans Visual Basic

Une *procédure* est un bloc d'instructions Visual Basic entouré par une instruction de déclaration (**Function**, **Sub**, **Operator**, **Get**, **Set**) et une déclaration **End** correspondante. Toutes les instructions exécutables en Visual Basic doivent être contenues dans une procédure.

Appel d'une procédure

Vous appelez une procédure à partir de n'importe quel emplacement dans le code. C'est ce qu'on appelle un *appel de procédure*. Lorsque la procédure a fini de s'exécuter, elle retourne le contrôle au code qui l'a appelé, en d'autres termes, le *code appelant*. Le code appelant est une instruction, ou une expression comprise dans une instruction, qui spécifie la procédure par son nom et lui transfère le contrôle.

Retour d'une procédure

Une procédure retourne le contrôle au code appelant à la fin de son exécution. Pour ce faire, elle peut utiliser une instruction Return, instruction (Visual Basic), l'instruction Exit, instruction (Visual Basic) appropriée pour la procédure, ou l'instruction End (Visual Basic) de la procédure. Le contrôle passe ensuite au code appelant qui suit le point de l'appel de procédure.

- Avec une instruction **Return**, le contrôle retourne immédiatement au code appelant. Les instructions qui suivent l'instruction **Return** ne s'exécutent pas. Une même procédure peut contenir plusieurs instructions **Return**.
- Avec une instruction **Exit Sub** ou **Exit Function**, le contrôle retourne immédiatement au code appelant. Les instructions qui suivent l'instruction **Exit** ne s'exécutent pas. Une même procédure peut contenir plusieurs instructions **Exit** et vous pouvez mixer les instructions **Return** et **Exit** dans la même procédure.
- Si une procédure ne contient pas d'instructions **Return** ou **Exit**, elle conclut par une instruction **End Sub** ou **End Function**, **End Get** ou **End Set** qui suit la dernière instruction du corps de la procédure. L'instruction **End** retourne immédiatement le contrôle au code appelant. Une procédure peut contenir uniquement une instruction **End**.

Paramètres et arguments

Dans la plupart des cas, une procédure doit utiliser des données différentes à chaque appel. Vous pouvez passer ces informations à la procédure dans le cadre de l'appel de procédure. La procédure définit zéro ou plus de *paramètres*, chacun représentant une valeur que vous devez lui passer. À chaque paramètre dans la définition de procédure correspond un *argument* dans l'appel de procédure. Un argument représente la valeur que vous passez au paramètre correspondant dans un appel de procédure donné.

Types de procédures

Visual Basic utilise plusieurs types de procédures :

- Les procédures Sub, procédures exécutent des actions mais ne retournent aucune valeur au code d'appel.
- Les procédures de gestion des événements sont des procédures **Sub** qui s'exécutent en réponse à un événement déclenché par l'action d'un utilisateur ou une occurrence dans un programme.
- Les procédures Function, procédures retournent une valeur au code d'appel. Elles peuvent exécuter d'autres actions avant de retourner.
- Les procédures Procédures Property retournent et assignent des valeurs de propriétés aux objets ou aux modules.
- Les procédures Procédures d'opérateur définissent le comportement d'un opérateur standard lorsque l'un ou les deux opérandes désignent une classe ou une structure définie récemment.
- Les procédures Procédures génériques dans Visual Basic définissent un ou plusieurs *paramètres de type*, en plus de leurs paramètres normaux. Par conséquent, le code appelant peut passer des types de données spécifiques à chaque appel.

Procédures et code structuré

Chaque ligne de code exécutable dans votre application doit faire partie d'une procédure, par exemple `Main`, `calculate` ou `Button1_Click`. Si vous subdivisez les procédures de grande taille en procédures plus petites, vous facilitez la lecture de votre application.

Les procédures permettent d'exécuter des tâches répétées ou partagées, par exemple des calculs fréquents, des manipulations

de textes et de contrôles ou des opérations de base de données. Vous pouvez appeler une procédure à partir de divers emplacements dans votre code, ce qui vous permet d'utiliser les procédures comme des blocs de construction pour votre application.

Le fait de structurer votre code à l'aide de procédures vous apporte les avantages suivants :

- Les procédures vous permettent de décomposer vos programmes en unités logiques discrètes. Vous pouvez déboguer plus facilement des unités séparées qu'un programme entier sans procédures.
- Vous pouvez utiliser des procédures développées pour un programme dans d'autres programmes, avec peu ou pas de modifications. Cela permet d'éviter la duplication du code.

Voir aussi

Tâches

Comment : créer une procédure

Concepts

Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure
Procédures récursives
Surcharge de procédure
Procédures génériques dans Visual Basic

Autres ressources

Programmation orientée objet dans Visual Basic

Comment : fractionner des grands segments de code en segments plus petits

Toutes les instructions exécutables en Visual Basic doivent être à l'intérieur d'une procédure telle que `Main`, `Form1_Load` ou `calculateTotal`. Vous pouvez écrire toute votre application sous la forme d'une seule procédure de grande taille, mais si vous la divisez en plusieurs procédures plus petites, votre code est plus lisible.

La programmation structurée est une approche qui accentue la modularité des programmes et une structure hiérarchique au sein d'une application. En Visual Basic, la manière la plus simple d'obtenir une programmation structurée est d'utiliser judicieusement des procédures afin de diviser votre application en unités logiques discrètes. Vous pouvez déboguer chaque unité plus facilement que le programme entier. Vous pouvez également utiliser une procédure développée pour un programme dans d'autres programmes, souvent avec peu ou pas de modifications.

Division d'une procédure de grande taille

Pour diviser une procédure de grande taille en parties autonomes

1. Identifiez une ou plusieurs sections autonomes de votre code.
2. Pour chaque section autonome, déplacez le code source hors de la procédure de grande taille et entourez-le avec les instructions **Sub** et **End Sub**
3. À l'endroit dans votre procédure de grande taille où vous avez supprimé la section de code, ajoutez une instruction qui appelle la procédure **Sub**.

Retour d'une valeur à une procédure de grande taille

S'il est utile de retourner une valeur à la procédure de grande taille pour la nouvelle procédure, vous pouvez définir une procédure **Function**.

Décomposition d'une section qui retourne une valeur

1. Entourez le code source supprimé avec les instructions **Function** et **End Function** plutôt que **Sub** et **End Sub**.
2. À l'endroit où la procédure **Function** a la valeur indiquant qu'elle est prête à retourner au code appelant, ajoutez une instruction **Return**.
3. À l'endroit dans votre procédure de grande taille où vous avez supprimé la section de code, assurez-vous que l'instruction appelante effectue une opération à l'aide de la valeur retournée. Vous pouvez la stocker dans une variable ou l'utiliser dans une expression.

Voir aussi

Tâches

Comment : créer une procédure

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure
Procédures récursives
Surcharge de procédure

Autres ressources

Programmation orientée objet dans Visual Basic

Comment : créer une procédure

Vous insérez une procédure entre une instruction de déclaration initiale (**Sub** ou **Function**) et une instruction de la déclaration de la fin (**End Sub** ou **End Function**). Le code de la procédure se trouve entre ces instructions.

Une procédure ne peut pas contenir une autre procédure, ses instructions de début et de fin doivent donc se situer en dehors de toute autre procédure.

Si un code effectue la même tâche dans des endroits différents, vous pouvez écrire la tâche une fois en tant que procédure puis l'appeler depuis des endroits différents dans votre code.

Pour créer une procédure qui ne retourne pas de valeur

1. En dehors de toute autre procédure, utilisez une instruction **Sub**, suivie d'une instruction **End Sub**
2. Dans l'instruction **Sub**, faites suivre le mot clé **Sub** du nom de la procédure, puis de la liste de paramètres entre parenthèses.
3. Placez les instructions de code de la procédure entre les instructions **Sub** et **End Sub**

Pour créer une procédure qui retourne une valeur

1. En dehors de toute autre procédure, utilisez une instruction **Function**, suivie d'une instruction **End Function**
2. Dans l'instruction **Function**, faites suivre le mot clé **Function** du nom de la procédure, puis de la liste de paramètres entre parenthèses, puis d'une clause **As** spécifiant le type de données de la valeur de retour.
3. Placez les instructions de code de la procédure entre les instructions **Function** et **End Function**
4. Utilisez une instruction **Return** pour retourner la valeur au code appelant.

Pour connecter votre nouvelle procédure avec les anciens blocs de code répétitifs

1. Assurez-vous que vous définissez la nouvelle procédure dans un endroit auquel l'ancien code a accès.
2. Dans votre ancien bloc de code répétitif, remplacez les instructions qui effectuent la tâche répétitive par une seule instruction qui appelle la procédure **Sub** ou **Function**.
3. Si votre procédure est une **Function** qui retourne une valeur, assurez-vous que votre instruction appelante exécute une action avec la valeur retournée, telle que le stockage dans une variable, dans le cas contraire la valeur sera perdue.

Exemple

La procédure **Function** suivante calcule le côté le plus long, ou hypoténuse, d'un triangle rectangle, d'après les valeurs des deux autres côtés.

VB

```
Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
    Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
End Function
```

Voir aussi

Tâches

Comment : fractionner des grands segments de code en segments plus petits

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure
Procédures récursives
Surcharge de procédure

Autres ressources

Programmation orientée objet dans Visual Basic

Sub, procédures

Une procédure **Sub** est une série d'instructions Visual Basic délimitées par les instructions **Sub** et **End Sub**. La procédure **Sub** effectue une tâche, puis retourne le contrôle au code appelant, mais ne retourne pas de valeur au code appelant.

Chaque fois que la procédure est appelée, ses instructions sont exécutées à partir de la première instruction exécutable située après l'instruction **Sub** jusqu'à la première instruction **End Sub**, **Exit Sub** ou **Return** rencontrée.

Vous pouvez définir une procédure **Sub** dans les modules, les classes et les structures. La valeur par défaut est **Public**, ce qui signifie que vous pouvez l'appeler de n'importe où dans votre application qui a accès au module, à la classe ou à la structure dans laquelle vous l'avez définie. Le terme, *méthode*, décrit une procédure **Sub** ou **Function**, à laquelle on peut accéder en dehors de son module, de sa classe ou de sa structure de définition. Pour plus d'informations, consultez Méthodes de classe.

Une procédure **Sub** peut accepter des arguments tels que les constantes, les variables ou les expressions qui sont passées à la procédure par le code appelant.

Syntaxe de déclaration

La syntaxe de déclaration d'une procédure **Sub** est la suivante :

```
[ modifiers ] Sub subname [( parameterlist )]
' Statements of the Sub procedure.
End Sub
```

modifiers peuvent spécifier un niveau d'accès et des informations concernant la surcharge, la substitution, le partage, et l'occultation. Pour plus d'informations, consultez Sub, instruction (Visual Basic).

Déclaration de paramètre

Déclarez chaque paramètre de procédure de la même manière que lorsque vous déclarez une variable, en spécifiant le nom de paramètre et le type de données. Vous pouvez également spécifier le mécanisme de passage et indiquer si le paramètre est facultatif ou s'il s'agit d'un tableau de paramètres.

La syntaxe de chaque paramètre dans la liste des paramètres est la suivante :

```
[Optional] [ByVal | ByRef] [ParamArray] parametername As datatype
```

Si le paramètre est facultatif, vous devez également fournir une valeur par défaut dans le cadre de sa déclaration. La syntaxe de spécification d'une valeur par défaut est la suivante :

```
Optional [ByVal | ByRef] parametername As datatype = defaultvalue
```

Paramètres en tant que variables locales

Lorsque contrôle passe à la procédure, chaque paramètre est traité comme une variable locale. Cela signifie que sa durée de vie est la même que celle de la procédure et que sa portée est la procédure entière.

Syntaxe d'appel

Vous appelez une procédure **Sub** de manière explicite à l'aide d'une instruction d'appel autonome. Vous ne pouvez pas l'appeler en utilisant son nom dans une expression. Vous devez fournir les valeurs de tous les arguments qui ne sont pas facultatifs et mettre la liste des arguments entre parenthèses. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses. L'utilisation du mot clé **Call** est facultative, mais non recommandée.

La syntaxe d'appel à une procédure **Sub** est la suivante :

```
[Call] subname [( argumentlist )]
```

Vous pouvez appeler une méthode **Sub** en dehors de la classe qui la définit. En premier lieu, vous devez utiliser le mot clé **New** pour créer une instance de la classe ou appeler une méthode qui retourne une instance de la classe. Pour plus d'informations, consultez Comment : utiliser le nouveau mot clé. Puis, vous pouvez utiliser la syntaxe suivante pour appeler la méthode **Sub** l'objet d'instance :

Object.methodname`[(`*argumentlist*`)]`

Pour plus d'informations, consultez Comment : exécuter des actions avec les méthodes.

Illustration de déclaration et d'appel

La procédure **Sub** ci-dessous indique à l'opérateur la tâche que l'application est sur le point d'exécuter, et affiche également des informations de date. Au lieu de dupliquer ce code au début de chaque tâche, l'application appelle simplement `tellOperator` depuis différents emplacements. Chaque appel passe une chaîne dans l'argument `task` qui identifie la tâche en cours de démarrage.

VB

```
Sub tellOperator(ByVal task As String)
    Dim stamp As Date
    stamp = TimeOfDay()
    MsgBox("Starting " & task & " at " & CStr(stamp))
End Sub
```

L'exemple suivant montre un appel typique à `tellOperator`

VB

```
Call tellOperator("file update")
```

Voir aussi

Tâches

Comment : appeler une procédure qui ne retourne pas de valeur

Comment : appeler un gestionnaire d'événements en Visual Basic

Référence

Sub, instruction (Visual Basic)

Concepts

Procédures dans Visual Basic

Function, procédures

Procédures Property

Procédures d'opérateur

Paramètres et arguments d'une procédure

Comment : appeler une procédure qui ne retourne pas de valeur

La procédure **Sub** ne retourne pas de valeur au code appelant. Vous appelez cette procédure de manière explicite à l'aide d'une instruction d'appel autonome. Vous ne pouvez pas l'appeler en utilisant uniquement son nom dans une expression.

L'instruction d'appel peut éventuellement utiliser le mot clé **Call**. Cette action est recommandée pour améliorer la lisibilité de votre code.

Pour appeler une procédure Sub

1. Utilisez une instruction **Call**, en tapant le nom de la procédure **Sub** avant le mot clé **Call**.
2. Insérez des parenthèses après le nom de la procédure pour encadrer la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses. Toutefois les parenthèses simplifient la lecture de votre code.
3. Placez les arguments dans la liste des arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la procédure **Sub**.

   L'exemple suivant appelle la fonction Visual Basic AppActivate pour activer une fenêtre d'application. **AppActivate** considère le titre de la fenêtre comme unique argument. Elle ne retourne pas de valeur au code appelant. Si un processus Notepad n'est pas en cours d'exécution, l'exemple lève un ArgumentException. La procédure **Shell** suppose que les applications se trouvent dans les chemins d'accès spécifiés.

   VB

```
Dim notepadID As Integer
' Activate a running Notepad process.
AppActivate("Untitled - Notepad")
' AppActivate can also use the return value of the Shell function.
' Shell runs a new instance of Notepad.
notepadID = Shell("C:\WINNT\NOTEPAD.EXE", AppWinStyle.NormalFocus)
' Activate the new instance of Notepad.
AppActivate(notepadID)
```

Voir aussi

Tâches

Comment : créer une procédure
Comment : appeler une procédure qui retourne une valeur
Comment : appeler un gestionnaire d'événements en Visual Basic

Référence

Sub, instruction (Visual Basic)
Shell, fonction
ArgumentException

Concepts

Procédures dans Visual Basic
Sub, procédures
Paramètres et arguments d'une procédure

Comment : appeler un gestionnaire d'événements en Visual Basic

Un *événement* est une action ou une occurrence (par exemple un clic de souris ou le dépassement d'un seuil de crédit) reconnue par un composant de programme pour laquelle vous pouvez écrire du code en réponse. Un *gestionnaire d'événements* est le code que vous écrivez pour répondre à un événement.

Un gestionnaire d'événements en Visual Basic est une procédure **Sub**. Toutefois, vous ne l'appelez normalement pas de la même manière que d'autres procédures **Sub**. À la place, vous identifiez la procédure comme un gestionnaire pour l'événement. Pour cela, vous pouvez utiliser soit avec une clause Handles et une variableWithEvents, soit une instruction AddHandler. L'utilisation d'une clause **Handles** est la manière par défaut de déclarer un gestionnaire d'événements en Visual Basic. Les gestionnaires d'événements sont écrits de cette manière par les concepteurs lorsque vous programmez dans l'environnement de développement intégré (IDE). L'instruction **AddHandler** convient pour déclencher dynamiquement des événements au moment de l'exécution.

Lorsque l'événement se produit, Visual Basic appelle automatiquement la procédure de gestionnaire d'événements. Tout code qui a accès à l'événement peut faire en sorte qu'il se produise en exécutant une instruction RaiseEvent.

Vous pouvez associer plusieurs gestionnaires d'événements au même événement. Dans quelques cas, vous pouvez dissocier un gestionnaire pour un événement. Pour plus d'informations, consultez Événements en Visual Basic.

Pour appeler un gestionnaire d'événements à l'aide de Handles et de WithEvents

1. Assurez-vous que l'événement est déclaré avec une instruction Event.
2. Déclarez une variable objet au niveau du module ou de la classe, à l'aide du mot clé WithEvents. La clause **As** pour cette variable doit spécifier la classe qui déclenche l'événement.
3. Dans la déclaration de la procédure **Sub** de gestion des événements, ajoutez une clause Handles qui spécifie la variable **WithEvents** et le nom d'événement.
4. Lorsque l'événement se produit, Visual Basic appelle automatiquement la procédure **Sub**. Votre code peut utiliser une instruction **RaiseEvent** pour faire en sorte que l'événement se produise.

   L'exemple suivant définit un événement et une variable **WithEvents** qui font référence à la classe qui déclenche l'événement. La procédure **Sub** de gestion des événements utilise une clause **Handles** pour spécifier la classe et l'événement qu'elle gère.

   VB

```
Public Class raisesEvent
    Public Event somethingHappened()
    Dim WithEvents happenObj As New raisesEvent
    Public Sub processHappen() Handles happenObj.somethingHappened
        ' Insert code to handle somethingHappened event.
    End Sub
End Class
```

Pour appeler un gestionnaire d'événements à l'aide d'AddHandler

1. Assurez-vous que l'événement est déclaré avec une instruction **Event**.
2. Exécutez une instruction AddHandler pour connecter dynamiquement la procédure **Sub** de gestion des événements avec l'événement.
3. Lorsque l'événement se produit, Visual Basic appelle automatiquement la procédure **Sub**. Votre code peut utiliser une instruction **RaiseEvent** pour faire en sorte que l'événement se produise.

   L'exemple suivant définit une procédure **Sub** pour gérer l'événement Closing d'un formulaire. Il utilise ensuite l'instruction AddHandler pour associer la procédure `catchClose` en tant que gestionnaire d'événements pour **Closing**.

VB

```
' Place these procedures inside a Form class definition.
Private Sub catchClose(ByVal sender As Object, ByVal e As System.ComponentModel.Cancel
EventArgs)
    ' Insert code to deal with impending closure of this form.
End Sub
Public Sub formOpened()
    AddHandler Me.Closing, AddressOf catchClose
End Sub
```

Vous pouvez dissocier un gestionnaire d'événements d'un événement en exécutant l'instruction RemoveHandler.

Voir aussi

Tâches

Comment : créer une procédure

Comment : appeler une procédure qui ne retourne pas de valeur

Référence

Sub, instruction (Visual Basic)

AddressOf, opérateur

Concepts

Procédures dans Visual Basic

Sub, procédures

Function, procédures

Une procédure **Function** est une série d'instructions Visual Basic encadrées par les instructions **Function** et **End Function**. La procédure **Function** effectue une tâche puis retourne le contrôle au code appelant. Lorsqu'elle retourne le contrôle, elle retourne également une valeur au code appelant.

Chaque fois que la procédure est appelée, ses instructions sont exécutées à partir de la première instruction exécutable située après l'instruction **Function** jusqu'à la première instruction **End Function**, **Exit Function** ou **Return** rencontrée.

Vous pouvez définir une procédure **Function** dans un module, une classe ou une structure. La valeur par défaut est **Public**, ce qui signifie que vous pouvez l'appeler de n'importe où dans une application qui a accès au module, à la classe ou à la structure dans laquelle vous l'avez définie.

Une procédure **Function** peut accepter des arguments tels que les constantes, les variables ou les expressions qui sont passées à la procédure par le code appelant.

Syntaxe de déclaration

La syntaxe de déclaration d'une procédure **Function** est la suivante :

```
[ modifiers ] Function functionname [( parameterlist )] As returntype
' Statements of the Function procedure.
End Function
```

modifiers peuvent spécifier un niveau d'accès et des informations concernant la surcharge, la substitution, le partage et l'occultation. Pour plus d'informations, consultez Function, instruction (Visual Basic).

Vous déclarez chaque paramètre de la même façon que pour Sub, procédures.

Type de données

Chaque procédure **Function** possède un type de données, comme n'importe quelle variable. Ce type de données est spécifié par la clause **As** dans l'instruction **Function** et il détermine le type de données de la valeur que la fonction retourne au code appelant. Les exemples de déclarations suivants illustrent ce principe.

```
Function yesterday() As Date
End Function
Function findSqrt(ByVal radicand As Single) As Single
End Function
```

Pour plus d'informations, consultez « Éléments » dans Function, instruction (Visual Basic).

Retour de valeurs

La valeur qu'une procédure **Function** retourne au code appelant s'appelle une *valeur de retour*. La procédure retourne cette valeur selon l'une des deux méthodes suivantes :

- Elle assigne une valeur à son nom de fonction dans une ou plusieurs instructions de la procédure. Le contrôle ne retourne pas au programme appelant tant qu'une instruction **Exit Function** ou **End Function** n'a pas été exécutée. L'exemple suivant illustre ce comportement :

  ```
  Function functionname [( parameterlist )] As returntype
  ' The following statement does not transfer control back to the calling code.
  functionname = expression
  ' When control returns to the calling code, expression is the return value.
  End Function
  ```

- Il utilise l'instruction **Return** pour spécifier la valeur de retour et retourne immédiatement le contrôle au programme appelant. L'exemple suivant illustre ce comportement :

  ```
  Function functionname [( parameterlist )] As returntype
  ```

```
' The following statement immediately transfers control back to the calling code and returns the value
of expression .

Return expression

End Function
```

L'assignation de la valeur de retour au nom de la fonction permet à la procédure de conserver le contrôle jusqu'à ce que celui-ci rencontre une instruction **Exit Function** ou **End Function**. Vous pouvez ainsi assigner une valeur préliminaire et l'ajuster ultérieurement si nécessaire.

Pour plus d'informations, consultez « Valeur de retour » dans Function, instruction (Visual Basic).

Retour de tableaux

Si la procédure **Function** retourne un type de données de tableau, vous ne pouvez pas accéder aux éléments individuels du tableau dans la fonction. Si vous essayez néanmoins, le compilateur interprète cette tentative comme un appel récursif à la procédure. L'exemple suivant illustre ce comportement :

```
Function allOnes(ByVal n As Integer) As Integer()

For i As Integer = 1 To n - 1

' The following statement generates a COMPILER ERROR .

allOnes(i) = 1

Next i

' The following statement generates a COMPILER ERROR .

Return allOnes()

End Function
```

Dans l'exemple précédent, le compilateur interprète la tentative d'assignation `allOnes(i) = 1` comme un appel à `allOnes` sur le côté gauche d'une instruction d'assignation. Il interprète la tentative `Return allOnes()` comme un appel sans argument. Les deux instructions génèrent des erreurs du compilateur.

Syntaxe d'appel

Vous appelez une procédure **Function** en incluant son nom et ses arguments soit à droite d'une instruction d'assignation soit dans une expression. Vous devez fournir les valeurs de tous les arguments qui ne sont pas facultatifs et placer la liste des arguments entre parenthèses. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses.

La syntaxe d'appel à une procédure **Function** est la suivante :

lvalue = *functionname* `[(` *argumentlist* `)]`

`If ((`*functionname* `[(` *argumentlist* `)] / 3) <=` *expression* `) Then`

Lorsque vous appelez une procédure **Function**, vous n'avez pas besoin d'utiliser sa valeur de retour. Dans le cas contraire, toutes les actions de la fonction sont effectuées, mais la valeur de retour est ignorée. MsgBox est souvent appelé de cette manière.

Illustration de déclaration et d'appel

La procédure **Function** suivante calcule le côté le plus long, ou hypoténuse, d'un triangle rectangle, d'après les valeurs des deux autres côtés.

VB

```
Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
    Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
End Function
```

L'exemple suivant montre un appel typique à `hypotenuse`.

VB

```
Dim testLength, testHypotenuse As Single
testHypotenuse = hypotenuse(testLength, 10.7)
```

Fonctions Runtime de Visual Basic

Le runtime de Visual Basic fournit de nombreuses fonctions, situées dans l'espace de noms Microsoft.VisualBasic. Il fournit notamment les fonctions courantes, telles que la fonction Beep, la fonction MsgBox (Visual Basic) et la fonction StrComp (Visual Basic). Vous appelez ces fonctions de la même façon que vos procédures **Function**.

Voir aussi

Tâches

Comment : créer une procédure qui retourne une valeur
Comment : retourner une valeur d'une procédure
Comment : appeler une procédure qui retourne une valeur

Référence

Function, instruction (Visual Basic)

Concepts

Procédures dans Visual Basic
Sub, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure

Comment : créer une procédure qui retourne une valeur

Vous utilisez une procédure **Function** pour retourner une valeur au code appelant.

Pour créer une procédure qui retourne une valeur

1. En dehors de toute autre procédure, utilisez une instruction **Function**, suivie d'une instruction **End Function**
2. Dans l'instruction **Function**, faites suivre le mot clé **Function** du nom de la procédure, puis de la liste de paramètres entre parenthèses.
3. Faites suivre les parenthèses d'une clause **As** pour spécifier le type de données de la valeur retournée.
4. Placez les instructions de code de la procédure entre les instructions **Function** et **End Function**
5. Utilisez une instruction **Return** pour retourner la valeur au code appelant.

   La procédure **Function** suivante calcule le côté le plus long, ou hypoténuse, d'un triangle rectangle, d'après les valeurs des deux autres côtés.

   VB

   ```
   Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
       Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
   End Function
   ```

   L'exemple suivant montre un appel typique à `hypotenuse`.

   VB

   ```
   Dim testLength, testHypotenuse As Single
   testHypotenuse = hypotenuse(testLength, 10.7)
   ```

Voir aussi

Tâches

Comment : retourner une valeur d'une procédure
Comment : appeler une procédure qui retourne une valeur

Référence

Function, instruction (Visual Basic)

Concepts

Procédures dans Visual Basic
Sub, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure

Comment : retourner une valeur d'une procédure

Une procédure **Function** retourne une valeur au code appelant en exécutant une instruction **Return** ou en rencontrant une instruction **Exit Function** ou **End Function**.

Pour retourner une valeur à l'aide de l'instruction Return

1. Insérez une instruction **Return** au point où la tâche de la procédure est effectuée.
2. Faites suivre le mot clé **Return** d'une expression qui cède la valeur que vous souhaitez retourner au code appelant.
3. Une même procédure peut contenir plusieurs instructions **Return**.

   La procédure **Function** suivante calcule le côté le plus long, ou hypoténuse, d'un triangle rectangle et le retourne au code appelant.

   VB

   ```
   Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
       Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
   End Function
   ```

   L'exemple suivant présente un appel typique à `hypotenuse` qui stocke la valeur retournée.

   VB

   ```
   Dim testLength, testHypotenuse As Single
   testHypotenuse = hypotenuse(testLength, 10.7)
   ```

Pour retourner une valeur à l'aide de la fonction Exit ou End

1. Assignez une valeur au nom de la procédure à au moins un endroit dans la procédure **Function**.
2. Lorsque vous exécutez une instruction **Exit Function** ou **End Function**, Visual Basic retourne la valeur la plus récemment assignée au nom de la procédure.
3. Une même procédure peut contenir plusieurs instructions **Exit Function** et vous pouvez mixer les instructions **Return** et **Exit Function** dans la même procédure.
4. Une procédure **Function** peut contenir uniquement une instruction **End Function**.

   Pour plus d'informations, consultez « Valeur de retour » dans Function, instruction (Visual Basic).

Voir aussi

Tâches

Comment : créer une procédure qui retourne une valeur
Comment : appeler une procédure qui retourne une valeur

Référence

Function, instruction (Visual Basic)
Return, instruction (Visual Basic)

Concepts

Procédures dans Visual Basic
Sub, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure

Comment : appeler une procédure qui retourne une valeur

Une procédure **Function** retourne une valeur au code appelant. Vous l'appelez en incluant son nom et ses arguments à droite d'une instruction d'assignation ou dans une expression.

Vous pouvez utiliser le mot clé **Call** pour appeler une procédure **Function**. Toutefois, cette technique ignore la valeur retournée par la procédure.

Pour appeler une procédure Function dans une expression

1. Utilisez le nom de procédure **Function** de la même façon que vous utilisez une variable. Vous pouvez utiliser un appel de procédure **Function** n'importe où vous pouvez utiliser une variable ou une constante dans une expression.
2. Insérez des parenthèses après le nom de la procédure pour encadrer la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses. Toutefois les parenthèses simplifient la lecture de votre code.
3. Placez les arguments dans la liste d'arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la procédure **Function**.

   Vous pouvez également passer un ou plusieurs arguments par nom. Pour plus d'informations, consultez Passage des arguments par position et par nom.
4. La valeur retournée par la procédure participe à l'expression comme la valeur d'une variable ou d'une constante le ferait.

Pour appeler une procédure Function dans une instruction d'assignation

1. Utilisez le nom de procédure **Function** après le signe égal (**=**) dans l'instruction d'assignation.
2. Insérez des parenthèses après le nom de la procédure pour encadrer la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses. Toutefois les parenthèses simplifient la lecture de votre code.
3. Placez les arguments dans la liste d'arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la procédure **Function**, à moins que vous les passiez par nom.
4. La valeur retournée par la procédure est stockée dans la variable ou la propriété sur le côté gauche de l'instruction d'assignation.

Exemple

L'exemple suivant appelle la fonction Environ de Visual Basic pour récupérer la valeur d'une variable d'environnement du système d'exploitation. La première ligne appelle **Environ** dans une expression et la deuxième ligne l'appelle dans une instruction d'assignation. **Environ** prend le nom de variable comme unique argument. Il retourne la valeur de la variable au code appelant.

VB

```
MsgBox("Value of PATH is " & Environ("PATH"))
Dim currentPath As String = Environ("PATH")
```

Voir aussi

Tâches

Comment : créer une procédure qui retourne une valeur
Comment : retourner une valeur d'une procédure
Comment : appeler une procédure qui ne retourne pas de valeur

Référence

Function, instruction (Visual Basic)

Concepts

Function, procédures
Paramètres et arguments d'une procédure

Procédures Property

Une procédure Property est une série d'instructions Visual Basic qui manipulent une propriété personnalisée d'un module, d'une classe ou d'une structure. Les procédures Property sont également appelées *accesseurs de propriété*.

Visual Basic prend en charge les procédures Property suivantes :

- Une procédure **Get** retourne la valeur d'une propriété. Elle est appelée lorsque vous accédez à la propriété dans une expression.
- Une procédure **Set** affecte une valeur à une propriété, y compris une référence d'objet. Elle est appelée lorsque vous assignez une valeur à la propriété.

Les procédures Property sont généralement définies par paires, à l'aide des instructions **Get** et **Set**, mais vous pouvez définir chaque procédure à part selon que la propriété est en lecture seule (instruction Get) ou en écriture seule (instruction Set (Visual Basic)).

Vous pouvez définir des propriétés dans les classes, les structures et les modules. Les propriétés sont **Public** par défaut, ce qui signifie que vous pouvez les appeler de n'importe où dans votre application qui peut accéder au conteneur de la propriété.

Pour une comparaison des propriétés et des variables, consultez Différences entre les propriétés et les variables en Visual Basic.

Syntaxe de déclaration

Une propriété est définie par un bloc de code contenu dans les instructions Property et**End Property**. À l'intérieur de ce bloc, chaque procédure Property s'affiche sous forme de bloc interne encadré par une instruction de déclaration (**Get** ou **Set**) et l'instruction de fin **End** correspondante.

La syntaxe de déclaration d'une propriété et de ses procédures est la suivante :

```
[Default] [modifiers] Property propertyname[(parameterlist)] As datatype

[accesslevel] Get

' Statements of the Get procedure.

' The following statement returns expression as the property's value.

Return expression

End Get

[accesslevel] Set[(ByVal newvalue As datatype)]

' Statements of the Set procedure.

' The following statement assigns newvalue as the property's value.

lvalue = newvalue

End Set

End Property
```

Les *modifiers* peuvent spécifier un niveau d'accès et des informations concernant la surcharge, la substitution, le partage et l'occultation, ainsi que spécifier si la propriété est en lecture seule ou en écriture seule. Les *access level* sur la procédure **Get** ou **Set** peuvent être de tout niveau plus restrictif que le niveau d'accès spécifié pour la propriété elle-même. Pour plus d'informations, consultez Property, instruction.

Type de données

Le type de données et le niveau d'accès principal d'une propriété sont définis dans l'instruction **Property**, et non dans les procédures Property. Une propriété peut avoir uniquement un type de données. Par exemple, vous ne pouvez pas définir une propriété pour stocker une valeur **Decimal** et récupérer une valeur **Double**.

Niveau d'accès

Toutefois, vous pouvez définir un niveau d'accès principal pour une propriété et restreindre encore plus le niveau d'accès dans l'une de ses procédures Property. Par exemple, vous pouvez définir une propriété **Public** puis définir une procédure **Private Set** La procédure **Get** reste **Public**. Vous pouvez modifier le niveau d'accès dans une seule des procédures d'une propriété et vous pouvez uniquement le rendre plus restrictif que le niveau d'accès principal. Pour plus d'informations, consultez Comment : déclarer une propriété avec des niveaux d'accès mixtes.

Déclaration de paramètre

Vous déclarez chaque paramètre de la même façon que pour Sub, procédures, à la différence que le mécanisme de passage doit être **ByVal**.

La syntaxe de chaque paramètre dans la liste des paramètres est la suivante :

```
[Optional] ByVal [ParamArray] parametername As datatype
```

Si le paramètre est facultatif, vous devez également fournir une valeur par défaut dans le cadre de sa déclaration. La syntaxe de spécification d'une valeur par défaut est la suivante :

```
Optional ByVal parametername As datatype = defaultvalue
```

Valeur de la propriété

Dans une procédure **Get**, la valeur de retour est fournie à l'expression appelante en tant que valeur de la propriété.

Dans une procédure **Set**, la nouvelle valeur de propriété est transférée au paramètre de l'instruction **Set**. Si vous déclarez explicitement un argument, vous devez le faire en utilisant le même type de données que la propriété. Si vous ne déclarez pas d'argument, le compilateur utilise l'argument implicite `Value` pour représenter la nouvelle valeur à assigner à la propriété.

Syntaxe d'appel

Vous appelez une procédure Property implicitement en faisant la référence à la propriété. Utilisez le nom de la propriété de la même façon que le nom d'une variable, sauf que vous devez fournir des valeurs pour tous les arguments non facultatifs et placer la liste des arguments entre parenthèses. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses.

La syntaxe d'un appel implicite à une procédure **Set** est la suivante :

```
propertyname[(argumentlist)] = expression
```

La syntaxe d'un appel implicite à une procédure **Get** est la suivante :

```
lvalue = propertyname[(argumentlist)]
```

```
Do While (propertyname[(argumentlist)] > expression)
```

Illustration de déclaration et d'appel

La propriété suivante stocke un nom complet comme deux noms constitutifs, le prénom et le nom. Lorsque le code appelant lit `fullName`, la procédure **Get** combine les deux noms constitutifs et retourne le nom complet. Lorsque le code appelant assigne un nouveau nom complet, la procédure **Set** tente de le décomposer en deux noms constitutifs. S'il ne trouve pas d'espace, il le stocke en tant que prénom.

VB

```
Dim firstName, lastName As String
Property fullName() As String
    Get
      If lastName = "" Then
          Return firstName
      Else
          Return firstName & " " & lastName
      End If

    End Get
    Set(ByVal Value As String)
        Dim space As Integer = Value.IndexOf(" ")
        If space < 0 Then
            firstName = Value
            lastName = ""
        Else
            firstName = Value.Substring(0, space)
            lastName = Value.Substring(space + 1)
        End If
    End Set
End Property
```

L'exemple suivant montre des appels typiques aux procédures Property de `fullName`.

VB

```
fullName = "MyFirstName MyLastName"
MsgBox(fullName)
```

Voir aussi

Tâches

Comment : créer une propriété
Comment : appeler une procédure de propriété
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : placer une valeur dans une propriété
Comment : obtenir une valeur d'une propriété

Concepts

Procédures dans Visual Basic
Function, procédures
Procédures d'opérateur
Paramètres et arguments d'une procédure
Différences entre les propriétés et les variables en Visual Basic

Différences entre les propriétés et les variables en Visual Basic

Les variables et propriétés représentent toutes deux des valeurs auxquelles vous pouvez accéder. Toutefois, elles sont différentes en termes de stockage et d'implémentation.

Variables

Une *variable* correspond directement à un emplacement de mémoire. Vous définissez une variable avec une instruction de déclaration unique. Une variable peut être une *variable locale*, définie à l'intérieur d'une procédure et disponible uniquement dans cette procédure, ou ce peut être une *variable membre*, définie par l'utilisateur dans un module, une classe ou une structure, mais pas à l'intérieur d'une procédure. Une variable membre porte également le nom de *champ*.

Propriétés

Une *propriété* est un élément de données défini sur un module, une classe ou une structure. Vous définissez une propriété avec un bloc de code placé entre les instructions **Property** et **End Property**. Le bloc de code contient une procédure **Get**, une procédure **Set**, ou les deux. Ces procédures sont nommées *procédures property* ou *accesseurs de propriété*. Outre la récupération ou le stockage de la valeur de la propriété, ils peuvent également effectuer des actions personnalisées, telles que la mise à jour d'un compteur d'accès.

Différences

Le tableau suivant présente certaines différences importantes entre les variables et les propriétés.

Point de différence	Variable	Propriété
Déclaration	Instruction de déclaration unique	Série d'instructions dans un bloc de code
Implémentation	Emplacement de stockage unique	Code exécutable (procédures property)
Stockage	Associé directement à la valeur de la variable	En général le stockage interne n'est pas disponible en dehors du module ou de la classe conteneur de la propriété. La valeur de propriété peut ou ne peut pas exister en tant qu'élément [1] stocké
Code exécutable	Aucun	Doit posséder au moins une procédure
Accès en lecture et en écriture	Lecture/écriture ou lecture seule	Lecture/écriture, lecture seule ou écriture seule
Actions personnalisées (en plus de l'acceptation ou du retour d'une valeur)	Impossible	Peut être exécuté dans le cadre de la définition ou de la récupération de la valeur d'une propriété

[1] contrairement à une variable, la valeur d'une propriété ne peut pas correspondre directement à un élément unique de stockage. Le stockage peut être fractionné par commodité ou sécurité, ou la valeur peut être stockée dans un formulaire chiffré. Dans ces cas, la procédure **Get** assemblerait les éléments ou déchiffrerait la valeur stockée, et la procédure **Set** chiffrerait la nouvelle valeur ou la diviserait dans le stockage constitutif. Une valeur de propriété peut être éphémère, comme l'heure du jour, dans ce cas la procédure **Get** la calcule à la volée chaque fois que vous accédez à la propriété.

Voir aussi

Tâches

Comment : créer une propriété
Comment : déclarer une propriété avec des niveaux d'accès mixtes
Comment : appeler une procédure de propriété
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : placer une valeur dans une propriété
Comment : obtenir une valeur d'une propriété

Référence

Property, instruction
Dim, instruction (Visual Basic)

Concepts

Procédures Property
Paramètres et arguments d'une procédure

Comment : créer une propriété

Entourez une définition de propriété entre une instruction **Property** et une instruction **End Property**. Dans cette définition, définissez une procédure **Get** ou une procédure **Set**, ou les deux. L'intégralité du code de cette propriété doit se trouver dans ces procédures.

La procédure **Get** récupère la valeur de la propriété et la propriété **Set** stocke une valeur. Si vous souhaitez spécifier un accès en lecture/écriture, vous devez définir les deux procédures. **Get** doit être défini pour une propriété en lecture seule, et **Set** pour une propriété en écriture seule.

Pour créer une propriété

1. En dehors d'une propriété ou d'une procédure, utilisez une instruction Property, suivie d'une instruction **End Property**.
2. Si la propriété prend des paramètres, faites suivre le mot clé **Property** du nom de la procédure, puis de la liste de paramètres entre parenthèses.
3. Faites suivre les parenthèses d'une clause **As** pour spécifier le type de données de la valeur de la propriété. Vous devez spécifier le type de données, même pour une propriété en écriture seule.
4. Ajoutez les procédures **Get** et **Set**, selon le cas. Reportez-vous aux instructions suivantes.

Pour créer une procédure Get qui récupère une valeur de propriété

1. Entre les instructions **Property** et **End Property**, insérez une instruction Get, suivie d'une instruction **End Get**. Il n'est pas nécessaire de définir de paramètre pour la procédure **Get**.
2. Placez les instructions de code pour récupérer la valeur de la propriété entre les instructions **Get** et **End Get** Ce code peut inclure d'autres calculs et des manipulations de données en plus de la génération et du retour de la valeur de la propriété.
3. Utilisez une instruction **Return** pour retourner la valeur de la propriété au code appelant.

Vous devez écrire une procédure **Get** pour une propriété en lecture-écriture et pour une propriété en lecture seule. Vous ne devez pas définir de procédure **Get** pour une propriété en écriture seule.

Pour créer une procédure Set qui écrit la valeur d'une propriété

1. Entre les instructions **Property** et **End Property**, écrivez une Set, instruction (Visual Basic), suivie d'une instruction **End Set**
2. Dans l'instruction **Set**, faites suivre le mot clé **Set** d'une liste de paramètres entre parenthèses. Cette liste de paramètres doit inclure au moins un paramètre de valeur pour la valeur transférée par le code appelant. Le nom par défaut pour ce paramètre de valeur est `Value`, mais vous pouvez utiliser un nom différent selon le cas. Le paramètre de valeur doit avoir le même type de données que la propriété elle-même.
3. Placez les instructions de code pour stocker une valeur dans la propriété entre les instructions **Set** et **End Set**. Ce code peut inclure d'autres calculs et des manipulations de données en plus de la validation et du stockage de la valeur de la propriété.
4. Utilisez le paramètre de la valeur pour accepter la valeur fournie par le code appelant. Vous pouvez stocker cette valeur directement dans une instruction d'assignation ou l'utiliser dans une expression pour calculer la valeur interne à stocker.

Vous devez écrire une procédure **Set** pour une propriété en lecture-écriture et pour une propriété en écriture seule. Vous ne devez pas définir de procédure **Set** pour une propriété en lecture seule.

Exemple

L'exemple suivant crée une propriété en lecture/écriture qui stocke un nom complet comme deux noms constitutifs, le prénom et le nom. Lorsque le code appelant lit `fullName`, la procédure **Get** combine les deux noms constitutifs et retourne le nom complet. Lorsque le code appelant assigne un nouveau nom complet, la procédure **Set** tente de le décomposer en deux noms constitutifs. S'il ne trouve pas d'espace, il le stocke en tant que prénom.

VB

```
Dim firstName, lastName As String
Property fullName() As String
    Get
      If lastName = "" Then
          Return firstName
      Else
          Return firstName & " " & lastName
      End If

    End Get
    Set(ByVal Value As String)
        Dim space As Integer = Value.IndexOf(" ")
        If space < 0 Then
            firstName = Value
            lastName = ""
        Else
            firstName = Value.Substring(0, space)
            lastName = Value.Substring(space + 1)
        End If
    End Set
End Property
```

L'exemple suivant montre des appels typiques aux procédures Property de `fullName`. Le premier appel définit la valeur de propriété et le deuxième appel la récupère.

VB

```
fullName = "MyFirstName MyLastName"
MsgBox(fullName)
```

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes
Comment : appeler une procédure de propriété
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : placer une valeur dans une propriété
Comment : obtenir une valeur d'une propriété

Concepts

Procédures dans Visual Basic
Procédures Property
Paramètres et arguments d'une procédure
Différences entre les propriétés et les variables en Visual Basic

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Si vous souhaitez que les procédures **Get** et **Set** sur une propriété aient différents niveaux d'accès, vous pouvez utiliser le niveau le plus permissif dans l'instruction **Property** et le niveau le plus restrictif dans l'instruction **Get** ou **Set**. Utilisez des niveaux d'accès mixtes sur une propriété si vous souhaitez que certaines parties du code puissent obtenir la valeur de la propriété et que d'autres parties du code puissent modifier la valeur.

Pour plus d'informations sur les niveaux d'accès, consultez Niveaux d'accès dans Visual Basic.

Pour déclarer une propriété avec des niveaux d'accès mixtes

1. Déclarez normalement la propriété et spécifiez le niveau d'accès le moins restrictif (tel que **Public**) dans l'instruction **Property**.
2. Déclarez la procédure **Get** ou **Set** spécifiant le niveau d'accès le plus restrictif (tel que **Friend**).
3. Ne spécifiez pas de niveau d'accès sur l'autre procédure de propriété. Le niveau d'accès déclaré dans l'instruction **Property** est supposé. Vous pouvez restreindre l'accès sur une seule des procédures de propriété.

   VB

   ```
   Public Class employee
       Private salaryValue As Double
       Protected Property salary() As Double
           Get
               Return salaryValue
           End Get
           Private Set(ByVal value As Double)
               salaryValue = value
           End Set
       End Property
   End Class
   ```

   Dans l'exemple précédent, la procédure **Get** a le même accès **Protected** que la propriété elle-même, alors que la procédure **Set** a un accès **Private**. Une classe dérivée de `employee` peut lire la valeur `salary`, mais seule la classe `employee` peut la définir.

Voir aussi

Tâches

Comment : créer une propriété
Comment : appeler une procédure de propriété
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : placer une valeur dans une propriété
Comment : obtenir une valeur d'une propriété

Référence

Property, instruction

Concepts

Procédures dans Visual Basic
Procédures Property
Paramètres et arguments d'une procédure
Différences entre les propriétés et les variables en Visual Basic

Comment : appeler une procédure de propriété

Appelez une procédure de propriété en stockant une valeur dans la propriété ou en récupérant sa valeur. Accédez à une propriété de la même manière que vous accédez à une variable.

La procédure **Set** de la propriété stocke une valeur, et sa procédure **Get** récupère la valeur. Toutefois n'appelez pas explicitement ces procédures par leur nom. Utilisez la propriété dans une instruction d'assignation ou dans une expression, comme si vous stockiez ou récupériez la valeur d'une variable. Visual Basic effectue les appels aux procédures de la propriété.

Pour appeler la procédure Get d'une propriété

1. Utilisez le nom de propriété dans une expression tout comme vous utiliseriez un nom de variable. Vous pouvez utiliser une propriété partout où vous pouvez utiliser une variable ou une constante.

   - ou -

   Utilisez le nom de propriété qui suit le signe égal (**=**) dans une instruction d'assignation.

   L'exemple suivant lit la valeur de la propriété Now, en appelant implicitement sa procédure **Get**.

   VB

   ```
   Dim ThisMoment As Date
   ' The following statement calls the Get procedure of the Visual Basic Now property.
   ThisMoment = Now
   ```

2. Si la propriété accepte des arguments, faites suivre le nom de propriété de parenthèses à l'intérieur desquelles vous joindrez la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses.

3. Placez les arguments dans la liste d'arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la propriété.

La valeur de la propriété participe à l'expression comme le ferait une variable ou une constante, ou bien elle est stockée dans la variable ou la propriété située sur le côté gauche de l'instruction d'assignation.

Pour appeler la procédure Set d'une propriété

1. Spécifiez le nom de la propriété sur le côté gauche d'une instruction d'assignation.

   L'exemple suivant définit la valeur de la propriété TimeOfDay, en appelant implicitement la procédure **Set**.

   VB

   ```
   ' The following statement calls the Set procedure of the Visual Basic TimeOfDay proper
   ty.
   TimeOfDay = #12:00:00 PM#
   ```

2. Si la propriété prend des arguments, faites suivre le nom de propriété de parenthèses à l'intérieur desquelles vous joindrez la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses.

3. Placez les arguments dans la liste d'arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la propriété.

La valeur générée sur le côté droit de l'instruction d'assignation est stockée dans la propriété.

Voir aussi

Tâches

Comment : créer une propriété
Comment : déclarer une propriété avec des niveaux d'accès mixtes
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : placer une valeur dans une propriété

Comment : obtenir une valeur d'une propriété

Référence

Property, instruction

Get, instruction

Set, instruction (Visual Basic)

Concepts

Procédures Property

Paramètres et arguments d'une procédure

Différences entre les propriétés et les variables en Visual Basic

Comment : déclarer et appeler une propriété par défaut en Visual Basic

Une *propriété par défaut* est une classe ou une propriété de structure à laquelle votre code peut accéder sans la spécifier. Lorsque le code appelant nomme une classe ou une structure, mais pas une propriété et lorsque le contexte autorise l'accès à une propriété, Visual Basic résout l'accès à cette classe ou à la propriété par défaut de la structure s'il en existe une.

Une classe ou une structure peut avoir au plus une propriété par défaut. Toutefois, vous pouvez surcharger une propriété par défaut et avoir plusieurs versions de celle-ci.

Pour plus d'informations, consultez Propriétés par défaut.

Pour déclarer une propriété par défaut

1. Déclarez normalement la propriété. Ne spécifiez pas le mot clé **Shared** ou **Private**.
2. Incluez le mot clé **Default** dans la déclaration de propriété.
3. Spécifiez au moins un paramètre pour la propriété. Vous ne pouvez pas définir une propriété par défaut qui ne prend pas au moins un argument.

   VB

   ```
   Default Property myProperty(ByVal index As Integer) As String
   ```

Pour appeler une propriété par défaut

1. Déclarez une variable de la classe conteneur ou du type structure.

   VB

   ```
   Dim x As New class1(3)
   ```

2. Utilisez le nom de variable seul dans une expression où vous incluriez normalement le nom de propriété.

   VB

   ```
   MsgBox(x)
   ```

3. Faites suivre le nom de variable d'une liste d'arguments entre parenthèses. Une propriété par défaut doit prendre au moins un argument.

   VB

   ```
   MsgBox(x(1))
   ```

4. Pour récupérer la valeur de propriété par défaut, utilisez le nom de variable, avec une liste d'arguments, dans une expression ou à la suite du signe égal (=) dans une instruction d'assignation.

   VB

   ```
   MsgBox(x(1) & x(2) & x(3))
   ```

5. Pour définir la valeur de propriété par défaut, utilisez le nom de variable, avec une liste d'arguments, à gauche d'une instruction d'assignation.

VB

```
x(1) = "Hello"
x(2) = " "
x(3) = "World"
```

6. Vous pouvez toujours spécifier le nom de propriété par défaut avec le nom de variable, comme vous le feriez pour accéder à une autre propriété.

   VB

```
x.myProperty(1) = "Hello"
x.myProperty(2) = " "
x.myProperty(3) = "World"
```

Exemple

L'exemple suivant déclare une propriété par défaut dans une classe.

VB

```
Public Class class1
    Private myStrings() As String
    Sub New(ByVal size As Integer)
        ReDim myStrings(size)
    End Sub
    Default Property myProperty(ByVal index As Integer) As String
        Get
            ' The Get property procedure is called when the value
            ' of the property is retrieved.
            Return myStrings(index)
        End Get
        Set(ByVal Value As String)
            ' The Set property procedure is called when the value
            ' of the property is modified.
            ' The value to be assigned is passed in the argument
            ' to Set.
            myStrings(index) = Value
        End Set
    End Property
End Class
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

L'exemple suivant montre comment appeler la propriété par défaut `myProperty` sur la classe `class1`. Les trois instructions d'assignation stockent des valeurs dans `myProperty`, et l'appel MsgBox lit les valeurs.

VB

```
Sub Test()
    Dim x As New class1(3)
    x(1) = "Hello"
    x(2) = " "
    x(3) = "World"
    MsgBox(x(1) & x(2) & x(3))
End Sub
```

L'utilisation la plus commune d'une propriété par défaut est Item, propriété (objet Collection) sur plusieurs classes de collection.

Programmation fiable

Les propriétés par défaut peuvent entraîner une légère réduction des caractères de code source, mais elles peuvent rendre votre code plus difficile à lire. Si le code appelant ne connaît pas votre classe ou votre structure, lorsqu'il fait référence au nom de la classe ou de la structure, il ne peut pas déterminer avec certitude si cette référence accède à la classe ou à la structure elle-même ou à une propriété par défaut. Cela peut conduire à des erreurs du compilateur ou à des erreurs d'exécution logiques subtiles.

Vous pouvez légèrement réduire la probabilité que se produisent des erreurs de propriété par défaut en utilisant toujours l'instruction Option Strict pour affecter la valeur **On** au contrôle de type de compilateur.

Si vous envisagez d'utiliser une classe ou une structure prédéfinie dans votre code, vous devez déterminer si elle a une propriété par défaut, et, le cas échéant, indiquer son nom.

À cause de ces inconvénients, vous devez envisager de ne pas définir de propriétés par défaut. Afin de garantir la lisibilité du code, vous devez également envisager de toujours faire référence explicitement à toutes les propriétés, y compris aux propriétés par défaut.

Voir aussi

Tâches

Comment : créer une propriété
Comment : déclarer une propriété avec des niveaux d'accès mixtes
Comment : appeler une procédure de propriété
Comment : placer une valeur dans une propriété
Comment : obtenir une valeur d'une propriété

Référence

Property, instruction
Default (Visual Basic)

Concepts

Procédures Property
Paramètres et arguments d'une procédure
Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0
Différences entre les propriétés et les variables en Visual Basic

Comment : placer une valeur dans une propriété

Vous pouvez stocker une valeur dans une propriété en plaçant le nom de la propriété sur le côté gauche d'une instruction d'assignation.

La procédure **Set** de la propriété stocke une valeur, mais vous ne l'appelez pas explicitement par son nom. Vous utilisez la propriété tout comme vous utiliseriez une variable. Visual Basic se charge d'appeler les procédures de la propriété.

Pour stocker une valeur dans une propriété

1. Spécifiez le nom de la propriété sur le côté gauche d'une instruction d'assignation.

   L'exemple suivant affecte la valeur Midi à la propriété **TimeOfDay** de Visual Basic, en appelant sa procédure **Set** implicitement.

   VB

   ```
   ' The following statement calls the Set procedure of the Visual Basic TimeOfDay proper
   ty.
   TimeOfDay = #12:00:00 PM#
   ```

2. Si la propriété accepte des arguments, faites suivre le nom de propriété de parenthèses à l'intérieur desquelles vous joindrez la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses.

3. Placez les arguments dans la liste d'arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la propriété.

4. La valeur générée sur le côté droit de l'instruction d'assignation est stockée dans la propriété.

Voir aussi

Tâches

Comment : créer une propriété
Comment : déclarer une propriété avec des niveaux d'accès mixtes
Comment : appeler une procédure de propriété
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : obtenir une valeur d'une propriété

Référence

Property, instruction
TimeOfDay, propriété

Concepts

Procédures Property
Paramètres et arguments d'une procédure
Différences entre les propriétés et les variables en Visual Basic

Comment : obtenir une valeur d'une propriété

Vous pouvez récupérer la valeur d'une propriété en incluant le nom de propriété dans une expression.

La procédure **Get** de la propriété récupère la valeur, mais vous ne l'appelez pas explicitement par son nom. Vous utilisez la propriété tout comme vous utiliseriez une variable. Visual Basic se charge d'appeler les procédures de la propriété.

Pour récupérer une valeur à partir d'une propriété

1. Utilisez le nom de propriété dans une expression tout comme vous utiliseriez un nom de variable. Vous pouvez utiliser une propriété partout où vous pouvez utiliser une variable ou une constante.

   - ou -

   Utilisez le nom de propriété qui suit le signe égal (**=**) dans une instruction d'assignation.

   L'exemple suivant lit la valeur de la propriété **Now** de Visual Basic, en appelant sa procédure **Get** implicitement.

   VB

   ```
   Dim ThisMoment As Date
   ' The following statement calls the Get procedure of the Visual Basic Now property.
   ThisMoment = Now
   ```

2. Si la propriété accepte des arguments, faites suivre le nom de propriété de parenthèses à l'intérieur desquelles vous joindrez la liste d'arguments. Si aucun argument n'est spécifié, vous pouvez ne pas mettre les parenthèses.

3. Placez les arguments dans la liste d'arguments entre parenthèses en les séparant par des virgules. Assurez-vous de fournir les arguments dans le même ordre que les paramètres correspondants définis par la propriété.

La valeur de la propriété participe à l'expression comme le ferait une variable ou une constante, ou bien elle est stockée dans la variable ou la propriété située sur le côté gauche de l'instruction d'assignation.

Voir aussi

Tâches

Comment : créer une propriété
Comment : déclarer une propriété avec des niveaux d'accès mixtes
Comment : appeler une procédure de propriété
Comment : déclarer et appeler une propriété par défaut en Visual Basic
Comment : placer une valeur dans une propriété

Référence

Property, instruction

Concepts

Procédures dans Visual Basic
Procédures Property
Paramètres et arguments d'une procédure
Différences entre les propriétés et les variables en Visual Basic

Procédures d'opérateur

Une procédure d'opérateur est une série d'instructions Visual Basic qui définissent le comportement d'un opérateur standard (tel que *****, **<>** ou **And**) sur une classe ou structure définie. Elle est également appelée *surcharge d'opérateur*.

Quand définir des procédures d'opérateur

Lorsque vous avez défini une classe ou structure, vous pouvez déclarer des variables comme étant du type de cette classe ou structure. Une telle variable doit parfois participer à une opération intégrée à une expression. Pour ce faire, elle doit être un opérande d'un opérateur.

Visual Basic définit uniquement des opérateurs sur ses types de données fondamentaux. Vous pouvez définir le comportement d'un opérateur lorsqu'au moins un des opérandes est du type de votre classe ou structure.

Pour plus d'informations, consultez Operator, instruction.

Types de procédure d'opérateur

Une procédure d'opérateur peut être de l'un des types suivants :

- Une définition d'un opérateur unaire dans laquelle l'argument est du type de votre classe ou structure.
- Une définition d'un opérateur binaire dans lequel au moins un des arguments est du type de votre classe ou structure.
- Une définition d'un opérateur de conversion dans lequel l'argument est du type de votre classe ou structure.
- Une définition d'un opérateur de conversion qui retourne le type de votre classe ou structure.

Les opérateurs de conversion sont toujours unaires, et vous utilisez toujours **CType** comme opérateur en cours de définition.

Syntaxe de déclaration

La syntaxe de déclaration d'une procédure d'opérateur est la suivante :

```
Public Shared [Widening | Narrowing] Operator operatorsymbol ( operand1 [, operand2 ]) As datatype
' Statements of the operator procedure.
End Operator
```

Utilisez uniquement le mot clé **Widening** ou **Narrowing** sur un opérateur de conversion de type. Le symbole de l'opérateur est toujours CType, fonction pour un opérateur de conversion de type.

Vous déclarez deux opérandes pour définir un opérateur binaire, et un seul opérande pour définir un opérateur unaire, y compris un opérateur de conversion de type. Tous les opérandes doivent être déclarés comme étant **ByVal**.

Vous déclarez chaque opérande de la même manière que vous déclarez des paramètres pour Sub, procédures.

Type de données

Comme vous définissez un opérateur sur une classe ou structure que vous avez définie, au moins un des opérandes doit être du même type de données que cette classe ou structure. Pour un opérateur de conversion de type, l'opérande ou le type de retour doit être du même type de données que la classe ou structure.

Pour plus d'informations, consultez Operator, instruction.

Syntaxe d'appel

Vous devez appeler de manière implicite une procédure d'opérateur en utilisant le symbole d'opérateur dans une expression. Vous fournissez les opérandes de la même manière que pour les opérateurs prédéfinis.

La syntaxe d'un appel implicite à une procédure d'opérateur est la suivante :

```
Dim testStruct As structurename
Dim testNewStruct As structurename = testStruct operatorsymbol 10
```

Illustration de déclaration et d'appel

La structure suivante stocke une valeur entière 128 bits signée comme parties constitutives de poids fort et de poids faible. Elle définit l'opérateur **+** pour ajouter deux valeurs `veryLong` et générer une valeur `veryLong` résultante.

VB

```
Public Structure veryLong
    Dim highOrder As Long
    Dim lowOrder As Long
    Public Shared Operator +(ByVal v As veryLong, _
                             ByVal w As veryLong) As veryLong
        Dim sum As New veryLong
        sum = v
        Try
            sum.lowOrder += w.lowOrder
        Catch ex As System.OverflowException
            sum.lowOrder -= (Long.MaxValue - w.lowOrder + 1)
            sum.highOrder += 1
        End Try
        sum.highOrder += w.highOrder
        Return sum
    End Operator
End Structure
```

L'exemple suivant montre un appel typique à l'opérateur **+** dont la valeur est `veryLong`.

VB

```
Dim v1, v2, v3 As veryLong
v1.highOrder = 1
v1.lowOrder = Long.MaxValue
v2.highOrder = 0
v2.lowOrder = 4
v3 = v1 + v2
```

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion
Comment : appeler une procédure d'opérateur
Comment : utiliser une classe qui définit des opérateurs

Référence

Operator, instruction

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Paramètres et arguments d'une procédure

Comment : définir un opérateur

Si vous avez défini une classe ou une structure, vous pouvez définir le comportement d'un opérateur standard (tel que *****, **<>** ou **And**) lorsqu'au moins une des opérandes est du même type que votre classe ou structure.

Définissez l'opérateur standard comme une procédure d'opérateur dans la classe ou structure. Toutes les procédures d'opérateur doivent être **Public Shared**.

La définition d'un opérateur sur une classe ou une structure est également appelée *surcharge* de l'opérateur.

Exemple

L'exemple suivant définit l'opérateur **+** pour une structure appelée `height`. La structure utilise des hauteurs exprimées en pieds et pouces. Un *pouce* équivaut à 2,54 centimètres et un *pied* équivaut à 12 pouces. Pour garantir des valeurs normalisées (pouces < 12,0), le constructeur exécute une arithmétique *modulo* 12. L'opérateur **+** utilise le constructeur pour générer des valeurs normalisées.

VB

```
Public Shadows Structure height
    ' Need Shadows because System.Windows.Forms.Form also defines property Height.
    Private feet As Integer
    Private inches As Double
    Public Sub New(ByVal f As Integer, ByVal i As Double)
        Me.feet = f + (CInt(i) \ 12)
        Me.inches = i Mod 12.0
    End Sub
    Public Overloads Function ToString() As String
        Return Me.feet & "' " & Me.inches & """"
    End Function
    Public Shared Operator +(ByVal h1 As height, ByVal h2 As height) _
        As height
        Return New height(h1.feet + h2.feet, h1.inches + h2.inches)
    End Operator
End Structure
```

Vous pouvez tester la structure `height` à l'aide du code suivant.

VB

```
Public Sub consumeHeight()
    Dim p1 As New height(3, 10)
    Dim p2 As New height(4, 8)
    Dim p3 As height = p1 + p2
    Dim s As String = p1.ToString() & " + " & p2.ToString() _
        & " = " & p3.ToString() & " (= 8' 6"" ?)"
    Dim p4 As New height(2, 14)
    s &= vbCrLf & "2' 14"" = " & p4.ToString() & " (= 3' 2"" ?)"
    Dim p5 As New height(4, 24)
    s &= vbCrLf & "4' 24"" = " & p5.ToString() & " (= 6' 0"" ?)"
    MsgBox(s)
End Sub
```

Voir aussi

Tâches

Comment : définir un opérateur de conversion
Comment : appeler une procédure d'opérateur
Comment : utiliser une classe qui définit des opérateurs
Comment : déclarer une structure

Référence

Operator, instruction
Structure, instruction

Concepts

Procédures d'opérateur

Comment : définir un opérateur de conversion

Si vous avez défini une classe ou structure, vous pouvez définir un opérateur de conversion du type de votre classe ou structure en un autre type de données (tel que **Integer**, **Double** ou **String**).

Définissez la conversion de type comme procédure de la fonction CType dans la classe ou structure. Toutes les procédures de conversion doivent être **Public Shared** et chacune doit spécifier Widening ou Narrowing.

La définition d'un opérateur sur une classe ou structure est également appelée *surcharge* de l'opérateur.

Exemple

L'exemple suivant définit des opérateurs de conversion entre une structure appelée `digit` et un **Byte**.

VB

```
Public Structure digit
Private dig As Byte
    Public Sub New(ByVal b As Byte)
        If (b < 0 OrElse b > 9) Then Throw New _
            System.ArgumentException("Argument outside range for Byte")
        Me.dig = b
    End Sub
    Public Shared Widening Operator CType(ByVal d As digit) As Byte
        Return d.dig
    End Operator
    Public Shared Narrowing Operator CType(ByVal b As Byte) As digit
        Return New digit(b)
    End Operator
End Structure
```

Vous pouvez tester la structure `digit` à l'aide du code suivant.

VB

```
Public Sub consumeDigit()
    Dim d1 As New digit(4)
    Dim d2 As New digit(7)
    Dim d3 As digit = CType(CByte(3), digit)
    Dim s As String = "Initial 4 generates " & CStr(CType(d1, Byte)) _
        & vbCrLf & "Initial 7 generates " & CStr(CType(d2, Byte)) _
        & vbCrLf & "Converted 3 generates " & CStr(CType(d3, Byte))
    Try
        Dim d4 As digit
        d4 = CType(CType(d1, Byte) + CType(d2, Byte), digit)
    Catch e4 As System.Exception
        s &= vbCrLf & "4 + 7 generates " & """ & e4.Message & """
    End Try
    Try
        Dim d5 As digit = CType(CByte(10), digit)
    Catch e5 As System.Exception
        s &= vbCrLf & "Initial 10 generates " & """ & e5.Message & """
    End Try
    MsgBox(s)
End Sub
```

Voir aussi

Tâches

Comment : définir un opérateur
Comment : appeler une procédure d'opérateur
Comment : utiliser une classe qui définit des opérateurs
Comment : déclarer une structure

Référence

Operator, instruction
Structure, instruction

Concepts

Procédures d'opérateur
Conversions implicites et explicites
Conversions étendues et restrictives

Comment : appeler une procédure d'opérateur

Vous devez appeler une procédure d'opérateur en utilisant le symbole d'opérateur dans une expression. Dans le cas d'un opérateur de conversion, vous devez l'appeler en appelant la fonction CType afin de convertir une valeur d'un type de données en un autre.

Vous ne devez pas appeler pas de procédures d'opérateur explicitement. Vous devez simplement utiliser l'opérateur ou la fonction **CType** dans une instruction d'assignation ou une expression, de la même façon que lorsque vous utilisez normalement un opérateur. Visual Basic se charge d'appeler la procédure d'opérateur.

La définition d'un opérateur sur une classe ou une structure est également appelée *surcharge* de l'opérateur.

Pour appeler une procédure d'opérateur

1. Utilisez normalement le symbole d'opérateur dans une expression.
2. Assurez-vous que les types de données des opérandes sont appropriés pour l'opérateur et dans l'ordre correct.
3. L'opérateur contribue à la valeur de l'expression comme prévu.

Pour appeler une procédure d'opérateur de conversion

1. Utilisez **CType** à l'intérieur d'une expression.
2. Assurez-vous que les types de données des opérandes sont appropriés pour la conversion et dans l'ordre correct.
3. **CType** appelle la procédure d'opérateur de conversion et retourne la valeur convertie.

Exemple

L'exemple suivant crée deux structures TimeSpan, les ajoute et stocke le résultat dans une troisième structure **TimeSpan**. La structure **TimeSpan** définit des procédures d'opérateur pour surcharger plusieurs opérateurs standard.

VB

```
Dim firstSpan As New TimeSpan(3, 30, 0)
Dim secondSpan As New TimeSpan(1, 30, 30)
Dim combinedSpan As TimeSpan = firstSpan + secondSpan
Dim s As String = firstSpan.ToString() & _
          " + " & secondSpan.ToString() & _
          " = " & combinedSpan.ToString()
MsgBox(s)
```

Étant donné que **TimeSpan** surcharge l'opérateur **+** standard, l'exemple précédent appelle une procédure d'opérateur lorsqu'il calcule la valeur de `combinedSpan`.

Pour obtenir un exemple d'appel de procédure d'opérateur de conversation, consultez
Comment : utiliser une classe qui définit des opérateurs.

Compilation du code

Assurez-vous que la classe ou la structure que vous utilisez définit l'opérateur que vous souhaitez utiliser.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion
Comment : déclarer une structure

Référence

Operator, instruction
Widening
Narrowing
Structure, instruction

Concepts

Procédures d'opérateur

Conversions implicites et explicites
Conversions étendues et restrictives

Comment : utiliser une classe qui définit des opérateurs

Si vous utilisez une classe ou structure qui définit ses propres opérateurs, vous pouvez accéder à ces opérateurs depuis Visual Basic.

La définition d'un opérateur sur une classe ou structure est également appelée *surcharge* de l'opérateur.

Exemple

L'exemple suivant accède à la structure SQL SqlString qui définit les opérateurs de conversion (fonction CType) dans les deux directions entre une chaîne SQL et une chaîne Visual Basic. Utilisez **CType(***SQL string expression*, **String)** pour convertir une chaîne SQL en une chaîne Visual Basic et **CType(***Visual Basic string expression*, **SqlString)** pour convertir dans l'autre sens.

VB

```
' Insert the following line at the beginning of your source file.
Imports System.Data.SqlTypes
```

VB

```
Public Sub setJobString(ByVal g As Integer)
    Dim title As String
    Dim jobTitle As System.Data.SqlTypes.SqlString
    Select Case g
        Case 1
            title = "President"
        Case 2
            title = "Vice President"
        Case 3
            title = "Director"
        Case 4
            title = "Manager"
        Case Else
            title = "Worker"
    End Select
    jobTitle = CType(title, SqlString)
    MsgBox("Group " & CStr(g) & " generates title """ _
        & CType(jobTitle, String) & """")
End Sub
```

La structure **SqlString** passe un opérateur de conversion (fonction CType) de la valeur **String** à **SqlString** et un autre de la valeur **SqlString** à **String**. L'instruction qui assigne `title` `jobTitle` utilise le premier opérateur et l'appel MsgBox, fonction (Visual Basic) utilise le second.

Compilation du code

Assurez-vous que la classe ou la structure que vous utilisez définit l'opérateur que vous souhaitez utiliser. Ne partez pas du principe que la classe ou la structure a défini chaque opérateur comme disponible pour la surcharge. Pour obtenir la liste des opérateurs disponibles, consultez Operator, instruction.

Incluez l'instruction **Imports** appropriée pour la chaîne SQL au début de votre fichier source (dans le cas présent, System.Data.SqlTypes).

Votre projet doit avoir des références à System.Data et System.XML. Si vous devez les ajouter, consultez Comment : ajouter et supprimer des références dans Visual Studio (C#, J#).

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion
Comment : appeler une procédure d'opérateur
Comment : déclarer une structure

Référence

Widening
Narrowing
Structure, instruction

Concepts

Procédures d'opérateur
Conversions implicites et explicites
Conversions étendues et restrictives

Paramètres et arguments d'une procédure

Dans la plupart des cas, une procédure a besoin d'informations sur les circonstances dans lesquelles elle est appelée. Une procédure qui exécute des tâches répétitives ou partagées utilise des informations différentes pour chaque appel. Ces informations se composent des variables, constantes et expressions passées à la procédure lorsque celle-ci est appelée.

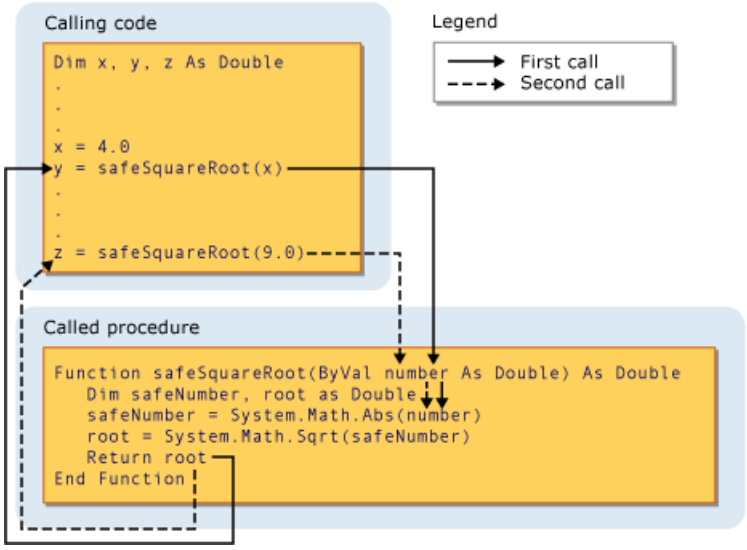
Un *paramètre* représente une valeur que vous êtes censé fournir lorsque vous appelez la procédure. La déclaration de la procédure définit ses paramètres.

Vous pouvez définir une procédure sans aucun paramètre, avec un seul paramètre ou avec plusieurs paramètres. La partie de la définition de procédure qui spécifie les paramètres est appelée *liste de paramètres*.

Un *argument* représente la valeur que vous fournissez à un paramètre de procédure lorsque vous appelez la procédure. Le code appelant fournit les arguments lorsqu'il appelle la procédure. La partie de l'appel de procédure qui spécifie les arguments est appelée *liste d'arguments*.

L'illustration suivante affiche le code appelant la procédure `safeSquareRoot` depuis deux places différentes. Le premier appel passe la valeur de la variable `x` (4.0) au paramètre `number` et la valeur de retour dans `root` (2.0) est assignée à la variable `y`. Le deuxième appel passe la valeur de littéral 9.0 à `number`, et assigne la valeur de retour (3.0) à la variable `z`.

Passage d'un argument à un paramètre



Pour plus d'informations, consultez Différences entre les paramètres et les arguments.

Type de données de paramètre

Définissez un type de données pour un paramètre en utilisant la clause **As** dans sa déclaration. Par exemple, la fonction suivante accepte une chaîne et un entier.

VB

```
Function appointment(ByVal day As String, ByVal hour As Integer) As String
    ' Insert code to return any appointment for the given day and time.
    Return "appointment"
End Function
```

Si le commutateur de vérification de type (Option Strict, instruction) est **Off,** la clause **As** est facultative, sauf si un paramètre l'utilise, dans quel cas tous les paramètres doivent l'utiliser. Si la vérification de type est **On**, la clause **As** est requise pour tous les paramètres de procédure.

Si le code appelant est censé fournir un argument avec un type de données différent de celui de son paramètre correspondant, tel que **Byte** à un paramètre **String**, il doit exécuter l'une des opérations suivantes :

- Fournissez uniquement des arguments avec des types de données qui s'étendent au type de données du paramètre ;
- Définissez **Option Strict Off** pour autoriser des conversions restrictives implicites ; ou

- Utilisez un mot clé de conversion pour convertir explicitement le type de données.

Paramètres de type

Une *procédure générique* définit également un ou plusieurs *paramètres de type* en plus de ses paramètres normaux. Une procédure générique autorise le code appelant à passer différents types de données à chaque appel de la procédure. Il peut ainsi adapter les types de données aux exigences de chaque appel. Consultez Procédures génériques dans Visual Basic.

Voir aussi

Tâches

Comment : définir un paramètre pour une procédure
Comment : passer des arguments à une procédure

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Passage d'arguments par valeur et par référence
Surcharge de procédure

Autres ressources

Conversions de type en Visual Basic

Différences entre les paramètres et les arguments

Dans la plupart des cas, une procédure a besoin d'informations sur les circonstances dans lesquelles elle est appelée. Une procédure qui exécute des tâches répétitives ou partagées utilise des informations différentes pour chaque appel. Ces informations se composent des variables, constantes et expressions passées à la procédure lorsque celle-ci est appelée.

Pour communiquer ces informations à la procédure, celle-ci définit un *paramètre*, et le code appelant passe un *argument* à ce paramètre. Vous pouvez comparer le paramètre à une place de parking et l'argument à une automobile. De même que plusieurs automobiles peuvent se garer sur la même place à différents instants, le code appelant peut passer un argument différent au même paramètre à chaque fois qu'il appelle la procédure.

Paramètres

Un *paramètre* représente une valeur que vous êtes censé passer lorsque vous appelez la procédure. La déclaration de la procédure définit ses paramètres.

Lorsque vous définissez une procédure **Function** ou **Sub**, vous spécifiez une *liste de paramètres* dans les parenthèses situées directement après le nom de la procédure. Pour chaque paramètre, spécifiez un nom, un type de données et un mécanisme de passage (ByVal ou ByRef). Vous pouvez également indiquer qu'un paramètre est facultatif, ce qui signifie que le code appelant n'est pas obligé de passer une valeur pour lui.

Le nom de chaque paramètre sert de *variable locale* dans la procédure. Vous utilisez le nom de paramètre de la même façon que vous utilisez toute autre variable.

Arguments

Un *argument* représente la valeur que vous passez à un paramètre de procédure lorsque vous appelez la procédure. Le code appelant fournit les arguments lorsqu'il appelle la procédure.

Lors d'un appel à une procédure **Function** ou **Sub**, incluez une *liste d'arguments* dans les parenthèses situées directement après le nom de la procédure. Chaque argument correspond au paramètre situé à la même position dans la liste.

Contrairement à une définition de paramètre, les arguments ne portent pas de noms. Chaque argument est une expression qui peut éventuellement contenir des variables, des constantes et des littéraux. Le type de données de l'expression évaluée doit correspondre normalement au type de données défini pour le paramètre correspondant, et dans tous les cas doit être convertible en le type de ce paramètre.

Voir aussi

Tâches

Comment : définir un paramètre pour une procédure
Comment : passer des arguments à une procédure

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Passage d'arguments par valeur et par référence
Procédures récursives
Surcharge de procédure

Comment : définir un paramètre pour une procédure

Un *paramètre* permet au code appelant de passer une valeur à la procédure lorsqu'il l'appelle. Vous devez déclarer chaque paramètre d'une procédure de la même façon qu'une variable, en spécifiant son nom et le type de données. Vous pouvez également spécifier le mécanisme de passage et indiquer si le paramètre est facultatif ou non.

Pour plus d'informations, consultez Paramètres et arguments d'une procédure.

Pour définir un paramètre de procédure

1. Dans la déclaration de procédure, ajoutez le nom de paramètre à la liste de paramètres de la procédure, en le séparant des autres paramètres par des virgules.
2. Décidez du type de données du paramètre.
3. Faites suivre le nom du paramètre d'une clause **As** qui spécifie le type de données.
4. Décidez du mécanisme de passage que vous souhaitez utiliser pour le paramètre. Normalement vous passez un paramètre par valeur, sauf si vous souhaitez que la procédure soit capable de modifier sa valeur dans le code appelant.
5. Faites précéder le nom de paramètre de ByVal ou ByRef pour spécifier le mécanisme de passage. Pour plus d'informations, consultez Différences entre le passage d'un argument par valeur et par référence.
6. Si le paramètre est facultatif, faites précéder le mécanisme de passage de Optional (Visual Basic) et faites suivre le type de données de paramètre d'un signe égal (**=**) et d'une valeur par défaut.

   L'exemple suivant définit le plan d'une procédure **Sub** avec trois paramètres. Les deux premiers sont requis et le troisième est facultatif. Les déclarations de paramètre sont séparées dans la liste de paramètres par des virgules.

   VB

   ```
   Sub updateCustomer(ByRef c As customer, ByVal region As String, _
     Optional ByVal level As Integer = 0)
     ' Insert code to update a customer object.
   End Sub
   ```

   Les premiers paramètres acceptent un objet `customer` et `updateCustomer` peut directement mettre à jour la variable passée à `c` parce que l'argument est passé ByRef. La procédure ne peut pas modifier les valeurs des deux derniers arguments parce qu'ils sont passés ByVal.

   Si le code appelant ne fournit pas de valeur pour le paramètre `level` , Visual Basic le définit à la valeur 0 par défaut.

   Si le commutateur de vérification de type (instruction Option Strict) a la valeur **Off**, la clause **As** est facultative lorsque vous définissez un paramètre. Toutefois, si un paramètre utilise une clause **As**, ils doivent tous l'utiliser. Si le commutateur de vérification de type est **On**, la clause **As** est requise pour chaque définition de paramètre.

   Le processus qui consiste à spécifier des types de données pour tous les éléments de programmation est appelé *typage fort*. Lorsque vous définissez **Option Strict On**, Visual Basic met en vigueur le typage fort. Cela est recommandé fortement, pour les raisons suivantes :

   - Cela permet la prise en charge de IntelliSens pour vos variables et vos paramètres. Cela vous permet de voir leurs propriétés et leurs autres membres à mesure que vous tapez le code.
   - Il permet au compilateur d'effectuer la vérification des types. Il permet d'intercepter les instructions susceptibles de provoquer, au moment de l'exécution, des erreurs telles qu'un dépassement de capacité. Il intercepte également les appels aux méthodes dans des objets qui ne les prennent pas en charge.
   - Il permet ainsi à votre code de s'exécuter plus rapidement. Une raison pour ceci est que si vous ne spécifiez pas de type de données pour un élément de programmation, le compilateur Visual Basic lui assigne le type **Object**. Votre code compilé peut avoir à passer de la conversion d'**Object** à celle d'autres types de données ce qui réduit la performance.

Voir aussi

Tâches

Comment : passer des arguments à une procédure

Concepts

Procédures dans Visual Basic

Sub, procédures

Function, procédures

Passage d'arguments par valeur et par référence

Procédures récursives

Surcharge de procédure

Autres ressources

Programmation orientée objet dans Visual Basic

Comment : passer des arguments à une procédure

Lorsque vous appelez une procédure, vous devez faire suivre le nom de procédure d'une liste d'arguments entre parenthèses. Vous devez fournir un argument qui correspond à chaque paramètre requis que la procédure définit et vous pouvez fournir facultativement des arguments aux paramètres **Optional**. Si vous ne fournissez pas de paramètre **Optional** lors de l'appel, vous devez inclure une virgule pour marquer sa place dans la liste d'arguments si vous fournissez des arguments suivants.

Si vous voulez passer un argument d'un type de données différent de celui du paramètre correspondant, tel que **Byte** à **String**, vous pouvez affecter la valeur **Off** au commutateur de vérification de type (instruction Option Strict). Si **Option Strict** a la valeur **On**, vous devez utiliser des conversions étendues ou des mots clés de conversion explicite. Pour plus d'informations, consultez Conversions étendues et restrictives et Fonctions de conversion de types de données.

Pour plus d'informations, consultez Paramètres et arguments d'une procédure.

Pour passer un ou plusieurs arguments à une procédure

1. Dans l'instruction appelante, faites suivre le nom de procédure de parenthèses.
2. À l'intérieur des parenthèses, placez une liste d'arguments. Incluez un argument pour chaque paramètre requis que la procédure définit et séparez les arguments avec des virgules.
3. Assurez-vous que chaque argument est une expression valide qui correspond à un type de données convertible au type que la procédure définit pour le paramètre correspondant.
4. Si un paramètre est défini comme Optional (Visual Basic), vous pouvez l'inclure dans la liste d'arguments ou l'omettre. Si vous l'omettez, la procédure utilise la valeur par défaut définie pour ce paramètre.
5. Si vous omettez un argument pour un paramètre **Optional** et s'il y a un autre paramètre après lui dans la liste de paramètres, vous pouvez marquer la place de l'argument omis par une virgule supplémentaire dans la liste d'arguments.

   L'exemple suivant appelle le MsgBox, fonction (Visual Basic) Visual Basic.

   VB

   ```
   Dim mbResult As MsgBoxResult
   Dim displayString As String = "Show this string to the user"
   mbResult = MsgBox(displayString, , "Put this in the title bar")
   ```

   L'exemple précédent fournit le premier argument requis qui représente la chaîne de message à afficher. Omet un argument pour le deuxième paramètre facultatif qui spécifie les boutons à afficher sur le message. Parce que l'appel ne fournit pas de valeur, **MsgBox** utilise la valeur par défaut, **MsgBoxStyle.OKOnly** qui affiche uniquement un bouton **OK**.

   La deuxième virgule dans la liste d'arguments marque la place du deuxième argument omis et la dernière chaîne est passée au troisième paramètre facultatif de **MsgBox** qui représente le texte à afficher dans la barre de titre.

Voir aussi

Tâches

Comment : définir un paramètre pour une procédure

Concepts

Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Passage d'arguments par valeur et par référence
Procédures récursives
Surcharge de procédure

Autres ressources

Programmation orientée objet dans Visual Basic

Passage d'arguments par valeur et par référence

En Visual Basic, vous pouvez passer un argument à une procédure *par valeur* ou *par référence*. Cette opération est appelée *mécanisme de passage* et détermine si la procédure peut modifier l'élément de programmation sous-jacent à l'argument dans le code appelant. La déclaration de procédure détermine le mécanisme de passage pour chaque paramètre en spécifiant le mot clé ByVal ou ByRef.

Distinctions

Lorsque vous passez un argument à une procédure, informez-vous des diverses distinctions qui ont une interaction entre elles :

- L'élément de programmation sous-jacent est-il modifiable ou non modifiable ?
- L'argument lui-même est-il modifiable ou non modifiable ?
- L'argument est-il passé par valeur ou par référence ?
- Le type de données d'argument est-il un type valeur ou un type référence ?

Pour plus d'informations, consultez Différences entre les arguments modifiables et non modifiables et Différences entre le passage d'un argument par valeur et par référence.

Choix du mécanisme de passage

Vous devez choisir avec soin le mécanisme de passage pour chaque argument.

- **Protection**. Lors du choix d'un mécanisme de passage, le critère le plus important est la fréquence à laquelle les variables d'appel doivent changer. Le passage d'un argument par **ByRef** permet à la procédure de retourner une valeur au code appelant via cet argument. Le passage d'un argument par **ByVal** permet d'empêcher une variable d'être modifiée par la procédure.
- **Performances**. Bien que le mécanisme de passage puisse affecter les performances de votre code, la différence est généralement insignifiante. La seule exception à cette règle est le passage d'un type valeur par **ByVal**. Dans ce cas, Visual Basic copie tout le contenu des données de l'argument. Par conséquent, dans le cas d'un type valeur de grande taille tel qu'une structure, il peut être préférable d'effectuer un passage **ByRef**.

  Pour les types référence, seul le pointeur vers les données est copié (quatre octets sur les plates-formes 32 bits, huit octets sur les plates-formes 64 bits). Par conséquent, vous pouvez passer des arguments de type **String** ou **Object** par valeur sans nuire aux performances.

Détermination du mécanisme de passage

La déclaration de procédure spécifie le mécanisme de passage pour chaque paramètre. Le code appelant ne peut pas substituer un mécanisme **ByVal**, mais si un argument est déclaré à l'aide de **ByRef**, le code appelant peut forcer le mécanisme à effectuer un passage **ByVal** en mettant le nom de l'argument entre parenthèses dans l'appel.

Le comportement par défaut en Visual Basic consiste à passer les arguments par valeur. Vous pouvez faciliter la lecture de votre code en utilisant le mot clé **ByVal**. En programmation, il est conseillé d'ajouter le mot clé **ByVal** ou **ByRef** à chaque paramètre déclaré.

Quand passer un argument par valeur

- Si l'élément de code appelant sous-jacent à l'argument est un élément non modifiable, déclarez le paramètre ByVal correspondant. Aucun code ne peut modifier la valeur d'un élément non modifiable.
- Si l'élément sous-jacent est modifiable, et si vous ne souhaitez pas que la procédure puisse modifier sa valeur, déclarez le paramètre **ByVal**. Seul le code appelant peut modifier la valeur d'un élément modifiable passée par valeur.

Quand passer un argument par référence

- Si la procédure nécessite véritablement que l'élément sous-jacent dans le code appelant soit modifié, déclarez le paramètre ByRef correspondant.
- Si l'exécution correcte du code dépend de la procédure modifiant l'élément sous-jacent dans le code appelant, déclarez le

paramètre **ByRef**. Si vous le passez par valeur ou si le code appelant substitue le mécanisme de passage **ByRef** en mettant l'argument entre parenthèses, l'appel de procédure peut produire des résultats inattendus.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : modifier la valeur d'un argument de la procédure
Comment : protéger un argument de procédure contre les modifications de valeur
Comment : forcer le passage d'un argument par valeur

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage des arguments par position et par nom
Types valeur et types référence

Différences entre les arguments modifiables et non modifiables

Lorsque vous appelez une procédure, vous lui passez généralement un ou plusieurs arguments. Chaque argument correspond à un élément de programmation sous-jacent. Les éléments sous-jacents et les arguments peuvent être modifiables ou non.

Éléments modifiables et non modifiables

Un élément de programmation peut être un *élément modifiable* dont la valeur peut changer ou un *élément non modifiable* dont la valeur reste fixe une fois qu'il est créé.

Le tableau suivant répertorie les éléments de programmation modifiables et non modifiables.

Éléments modifiables	Éléments non modifiables
Variables locales (déclarées à l'intérieur des procédures), comprenant les variables objet, sauf en lecture seule	Variables, champs et propriétés en lecture seule
Champs (variables membres de modules, de classes et de structures), sauf en lecture seule	Constantes et littéraux
Propriétés, sauf en lecture seule	Membres d'énumération
Éléments de tableau	Expressions (même si elles contiennent des éléments modifiables)

Arguments modifiables et non modifiables

Un *argument modifiable* est un élément comprenant un élément sous-jacent modifiable. Le code appelant peut stocker une nouvelle valeur à n'importe quel moment, et si vous passez l'argument ByRef, le code de la procédure peut également modifier l'élément sous-jacent du code appelant.

Un *argument non modifiable* possède un élément non modifiable sous-jacent ou est passé ByVal. La procédure ne peut pas modifier l'élément sous-jacent dans le code appelant, même si cet élément est modifiable. Si cet élément est non modifiable, le code appelant ne peut pas le modifier.

La procédure appelée peut modifier sa copie locale de l'argument non modifiable, mais cette modification n'affecte pas l'élément sous-jacent dans le code appelant.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : modifier la valeur d'un argument de la procédure
Comment : protéger un argument de procédure contre les modifications de valeur
Comment : forcer le passage d'un argument par valeur

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Différences entre le passage d'un argument par valeur et par référence
Passage des arguments par position et par nom
Types valeur et types référence

Différences entre le passage d'un argument par valeur et par référence

Lorsque vous passez un ou plusieurs arguments à une procédure, chaque argument correspond à un élément de programmation sous-jacent dans le code appelant. Vous pouvez passer la valeur de cet élément sous-jacent ou une référence à celui-ci. Ce concept porte le nom de *mécanisme de passage*.

Passage par valeur

Vous passez un argument *par valeur* en spécifiant le mot clé ByVal pour le paramètre correspondant dans la définition de la procédure. Lorsque vous utilisez ce mécanisme de passage, Visual Basic copie la valeur de l'élément de programmation sous-jacent dans une variable locale dans la procédure. Le code de la procédure n'a pas accès à l'élément sous-jacent dans le code appelant.

Passage par référence

Vous passez un argument *par référence* en spécifiant le mot clé ByRef pour le paramètre correspondant dans la définition de la procédure. Lorsque vous utilisez ce mécanisme de passage, Visual Basic fournit à la procédure une référence directe à l'élément de programmation sous-jacent dans le code appelant.

Mécanisme de passage et type d'élément

Le choix de mécanisme de passage est différent de la classification du type d'élément sous-jacent. Le passage par valeur ou par référence fait référence à ce que Visual Basic fournit au code de la procédure. Un type valeur ou un type référence fait référence à la manière dont un élément de programmation est stocké dans la mémoire.

Toutefois, le mécanisme de passage et le type d'élément sont étroitement liés. La valeur d'un type référence est un pointeur vers les données situées ailleurs dans la mémoire. Cela signifie que lorsque vous passez un type référence par valeur, le code de la procédure a un pointeur vers les données de l'élément sous-jacent, bien qu'il ne puisse pas accéder à l'élément sous-jacent lui-même. Par exemple, si l'élément est une variable tableau, le code de la procédure n'a pas accès à la variable elle-même, mais il peut accéder aux membres du tableau.

Possibilité de modifier

Lorsque vous passez un élément non modifiable en tant qu'argument, la procédure ne peut jamais le modifier dans le code appelant, qu'il soit passé **ByVal** ou **ByRef**.

Dans le cas d'un élément modifiable, le tableau suivant résume l'interaction entre le type de l'élément et le mécanisme de passage.

Type d'élément	Passé ByVal	Passé ByRef
Type valeur (ne contient qu'une valeur)	La procédure ne peut pas modifier la variable ni l'un de ses membres.	La procédure peut modifier la variable et ses membres.
Type référence (contient un pointeur vers une instance de classe ou de structure)	La procédure ne peut pas changer la variable, mais peut modifier les membres de l'instance qu'elle désigne.	La procédure peut changer la variable et les membres de l'instance qu'elle désigne.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : modifier la valeur d'un argument de la procédure
Comment : protéger un argument de procédure contre les modifications de valeur
Comment : forcer le passage d'un argument par valeur

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Différences entre les arguments modifiables et non modifiables
Passage des arguments par position et par nom
Types valeur et types référence

Comment : modifier la valeur d'un argument de la procédure

Lorsque vous appelez une procédure, chaque argument que vous fournissez correspond à l'un des paramètres définis dans la procédure. Dans certains cas, le code de procédure peut modifier la valeur sous-jacente à un argument dans le code appelant. Dans d'autres cas, la procédure peut modifier uniquement sa copie locale d'un argument.

Lorsque vous appelez la procédure, Visual Basic fait une copie locale de chaque argument auquel est passé ByVal. Pour chaque argument auquel est passé ByRef, Visual Basic donne au code de procédure une référence directe à l'élément de programmation sous-jacent à l'argument dans le code appelant.

Si l'élément sous-jacent dans le code appelant est un élément modifiable et si **ByRef** est passé à l'argument, le code de procédure peut utiliser la référence directe pour modifier la valeur de l'élément dans le code appelant.

Modification de la valeur sous-jacente

Pour modifier la valeur sous-jacente d'un argument de procédure dans le code appelant

1. Dans la déclaration de procédure, spécifiez ByRef pour le paramètre correspondant à l'argument.
2. Dans le code appelant, passez un élément de programmation modifiable comme argument.
3. Dans le code appelant, ne mettez pas l'argument entre parenthèses dans la liste d'arguments.
4. Dans le code de procédure, utilisez le nom de paramètre pour assigner une valeur à l'élément sous-jacent dans le code appelant.

Consultez l'exemple ci-dessous pour une démonstration.

Modification de copies locales

Si l'élément sous-jacent dans le code appelant est un élément non modifiable ou si **ByVal** est passé à l'argument, la procédure ne peut pas modifier sa valeur dans le code appelant. Toutefois, la procédure peut modifier sa copie locale d'un tel argument.

Pour modifier la copie d'un argument de procédure dans le code de procédure

1. Dans la déclaration de procédure, spécifiez ByVal pour le paramètre correspondant à l'argument.

   - ou -

   Dans le code appelant, mettez l'argument entre parenthèses dans la liste d'arguments. Cela force Visual Basic à passer l'argument par valeur, même si le paramètre correspondant spécifie **ByRef**.

2. Dans le code de procédure, utilisez le nom de paramètre pour assigner une valeur à la copie locale de l'argument. La valeur sous-jacente dans le code appelant n'est pas modifiée.

Exemple

L'exemple suivant illustre deux procédures qui acceptent une variable tableau et opèrent sur ses éléments. La procédure `increase` ajoute simplement la valeur 1 à chaque élément. La procédure `replace` assigne un nouveau tableau au paramètre `a()`, puis ajoute la valeur 1 à chaque élément.

VB

```
Public Sub increase(ByVal a() As Long)
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Public Sub replace(ByRef a() As Long)
    Dim k() As Long = {100, 200, 300}
    a = k
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
```

```
    Next j
End Sub
```

VB

```
Dim n() As Long = {10, 20, 30, 40}
Call increase(n)
MsgBox("After increase(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
Call replace(n)
MsgBox("After replace(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
```

Le premier appel **MsgBox** affiche "After increase(n): 11, 21, 31, 41". Étant donné que le tableau `n` est un type référence, `replace` peut modifier ses membres, même si le mécanisme de passage est **ByVal**.

Le deuxième appel **MsgBox** affiche "After replace(n): 101, 201, 301". Étant donné que **ByRef** est passé à `n`, `replace` peut modifier la variable `n` dans le code appelant et lui assigner un nouveau tableau. Étant donné que `n` est un type référence, `replace` peut également modifier ses membres.

Vous pouvez empêcher la procédure de modifier la variable elle-même dans le code appelant. Consultez Comment : protéger un argument de procédure contre les modifications de valeur.

Compilation du code

Lorsque vous passez une variable par référence, vous devez utiliser le mot clé **ByRef** pour spécifier ce mécanisme.

Le comportement par défaut en Visual Basic consiste à passer les arguments par valeur. En programmation, il est toutefois conseillé d'ajouter le mot clé ByVal ou ByRef avec chaque argument déclaré. Cela rend votre code plus lisible.

Sécurité

Permettre à une procédure de modifier la valeur sous-jacente à un argument dans le code appelant est toujours risqué. Vérifiez que cette valeur doit bien être modifiée et préparez-vous à en vérifier la validité avant de l'utiliser.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : protéger un argument de procédure contre les modifications de valeur
Comment : forcer le passage d'un argument par valeur

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Différences entre les arguments modifiables et non modifiables
Différences entre le passage d'un argument par valeur et par référence
Passage des arguments par position et par nom
Types valeur et types référence

Comment : protéger un argument de procédure contre les modifications de valeur

Si une procédure déclare un paramètre comme ByRef, Visual Basic donne au code de procédure une référence directe à l'élément de programmation sous-jacent à l'argument dans le code appelant. Cela permet à la procédure de modifier la valeur sous-jacente à l'argument dans le code appelant. Il peut arriver que le code appelant tente de se protéger contre une telle modification.

Vous pouvez toujours protéger un argument d'une modification en déclarant le paramètre ByVal correspondant dans la procédure. Si vous voulez avoir la possibilité de modifier un argument donné lorsque vous le souhaitez, vous pouvez le déclarer **ByRef** et laisser le code appelant déterminer le mécanisme de passage pour chaque appel. Il procède en mettant l'argument correspondant entre parenthèses pour le passer par valeur, ou en ne le mettant pas entre parenthèses pour le passer par référence. Pour plus d'informations, consultez Comment : forcer le passage d'un argument par valeur.

Exemple

L'exemple suivant illustre deux procédures qui acceptent une variable tableau et opèrent sur ses éléments. La procédure `increase` ajoute simplement la valeur 1 à chaque élément. La procédure `replace` assigne un nouveau tableau au paramètre `a()`, puis ajoute la valeur 1 à chaque élément. Cependant, la réassignation n'affecte pas la variable tableau sous-jacente dans le code appelant.

VB

```
Public Sub increase(ByVal a() As Long)
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Public Sub replace(ByVal a() As Long)
    Dim k() As Long = {100, 200, 300}
    a = k
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Dim n() As Long = {10, 20, 30, 40}
Call increase(n)
MsgBox("After increase(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
Call replace(n)
MsgBox("After replace(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
```

Le premier appel **MsgBox** affiche "After increase(n): 11, 21, 31, 41". Étant donné que le tableau `n` est un type référence, `replace` peut modifier ses membres, même si le mécanisme de passage est **ByVal**.

Le second appel **MsgBox** affiche "After replace(n): 11, 21, 31, 41". Étant donné que `n` est passé **ByVal**, `replace` ne peut pas modifier la variable `n` dans le code appelant en lui assignant un nouveau tableau. Lorsque `replace` crée la nouvelle instance de tableau `k` et l'assigne à la variable locale `a`, il perd la référence à `n` passée par le code appelant. Lorsqu'il modifie les membres de `a`, seul le tableau local `k` est affecté. Par conséquent, `replace` n'incrémente pas les valeurs du tableau `n` dans le code appelant.

Compilation du code

Le comportement par défaut en Visual Basic consiste à passer les arguments par valeur. En programmation, il est toutefois

conseillé d'ajouter le mot clé ByVal ou ByRef avec chaque argument déclaré. Cela rend votre code plus lisible.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : modifier la valeur d'un argument de la procédure
Comment : forcer le passage d'un argument par valeur

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Différences entre les arguments modifiables et non modifiables
Différences entre le passage d'un argument par valeur et par référence
Passage des arguments par position et par nom
Types valeur et types référence

Comment : forcer le passage d'un argument par valeur

La déclaration de procédure détermine le mécanisme de passage. Si un paramètre est déclaré ByRef, Visual Basic doit passer l'argument correspondant par référence. Cela permet à la procédure de modifier la valeur de l'élément de programmation sous-jacent à l'argument dans le code appelant. Si vous souhaitez protéger l'élément sous-jacent contre une telle modification, vous pouvez substituer le mécanisme de passage **ByRef** dans l'appel de procédure en plaçant le nom de l'argument entre parenthèses. Ces parenthèses s'ajoutent aux parenthèses qui encadrent la liste d'arguments dans l'appel.

Le code appelant ne peut pas substituer un mécanisme ByVal.

Pour forcer le passage d'un argument par valeur

- Si le paramètre correspondant est déclaré **ByVal** dans la procédure, aucune étape supplémentaire n'est requise. Visual Basic doit déjà passer l'argument par valeur.
- Si le paramètre correspondant est déclaré **ByRef** dans la procédure, placez l'argument entre parenthèses dans l'appel de procédure.

Exemple

L'exemple suivant substitue une déclaration de paramètre **ByRef**. Dans l'appel qui force **ByVal**, notez les deux niveaux de parenthèses.

VB

```
Sub setNewString(ByRef inString As String)
    inString = "This is a new value for the inString argument."
    MsgBox(inString)
End Sub
```

VB

```
Dim str As String = "Cannot be replaced if passed ByVal"

' The following call passes str ByVal even though it is declared ByRef.
Call setNewString((str))
' The parentheses around str protect it from change.
MsgBox(str)

' The following call allows str to be passed ByRef as declared.
Call setNewString(str)
' Variable str is not protected from change.
MsgBox(str)
```

Lorsque `str` est placé entre des parenthèses supplémentaires dans la liste d'arguments, la procédure `setNewString` ne peut pas modifier sa valeur dans le code appelant, et **MsgBox** affiche "Cannot be replaced if passed ByVal". Lorsque `str` n'est pas placé entre des parenthèses supplémentaires, la procédure peut le modifier, et **MsgBox** affiche "This is a new value for the inString argument".

Compilation du code

Lorsque vous passez une variable par référence, vous devez utiliser le mot clé **ByRef** pour spécifier ce mécanisme.

Le comportement par défaut en Visual Basic consiste à passer les arguments par valeur. En programmation, il est toutefois conseillé d'ajouter le mot clé ByVal ou ByRef avec chaque argument déclaré. Cela rend votre code plus lisible.

Programmation fiable

Si une procédure déclare un paramètre ByRef, l'exécution correcte du code peut dépendre de la capacité à modifier l'élément sous-jacent dans le code appelant. Si le code appelant substitue ce mécanisme appelant en plaçant l'argument entre parenthèses ou s'il passe un argument non modifiable, la procédure ne peut pas modifier l'élément sous-jacent. Cela peut produire des résultats inattendus dans le code appelant.

Sécurité

Permettre à une procédure de modifier la valeur sous-jacente à un argument dans le code appelant est toujours risqué. Vérifiez que cette valeur doit bien être modifiée et préparez-vous à en vérifier la validité avant de l'utiliser.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : modifier la valeur d'un argument de la procédure
Comment : protéger un argument de procédure contre les modifications de valeur

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Différences entre les arguments modifiables et non modifiables
Différences entre le passage d'un argument par valeur et par référence
Passage des arguments par position et par nom
Types valeur et types référence

Passage des arguments par position et par nom

Lorsque vous appelez une procédure **Sub** ou **Function**, vous pouvez passer des arguments *par position* (dans l'ordre où ils apparaissent dans la définition de la procédure) ou *par nom*, sans tenir compte de la position.

Lorsque vous passez un argument par nom, vous devez spécifier le nom déclaré de l'argument suivi de deux-points et d'un signe égal (**:=**), puis ajouter la valeur de l'argument. L'ordre des arguments nommés n'a pas d'importance.

Par exemple, la procédure **Sub** suivante accepte trois arguments :

VB

```
Sub studentInfo(ByVal name As String, _
       Optional ByVal age As Short = 0, _
       Optional ByVal birth As Date = #1/1/2000#)

  Debug.WriteLine("Name = " & name & _
                  "; age = " & CStr(age) & _
                  "; birth date = " & CStr(birth))
End Sub
```

Lorsque vous appelez cette procédure, vous pouvez fournir les arguments par position, par nom ou les deux.

Passage des arguments par position

Vous pouvez appeler la procédure `studentInfo` avec des arguments passés par position et séparés par des virgules, comme le montre l'exemple suivant :

VB

```
Call studentInfo("Mary", 19, #9/21/1981#)
```

Si vous omettez un argument facultatif dans une liste d'arguments positionnels, vous devez marquer son emplacement à l'aide d'une virgule. L'exemple suivant appelle `studentInfo` sans l'argument `age` :

VB

```
Call studentInfo("Mary", , #9/21/1981#)
```

Passage des arguments par nom

Vous pouvez également appeler `studentInfo` avec des arguments passés par nom et séparés par des virgules, comme le montre l'exemple suivant :

VB

```
Call studentInfo(age:=19, birth:=#9/21/1981#, name:="Mary")
```

Mélange d'arguments par position et par nom

Vous pouvez fournir des arguments à la fois par position et par nom dans un seul appel de procédure, comme le montre l'exemple suivant :

VB

```
Call studentInfo("Mary", birth:=#9/21/1981#)
```

Dans l'exemple précédent, aucune virgule supplémentaire n'est nécessaire pour marquer l'emplacement de l'argument `age` ayant été omis, car `birth` est passé par nom.

Lorsque vous fournissez des arguments à la fois par position et par nom, les arguments positionnels doivent tous être placés en premier. Une fois que vous avez fourni un argument par nom, les arguments restants doivent tous être fournis par nom.

Spécification d'arguments facultatifs par nom

Le fait de passer des arguments par nom est très utile lorsque vous appelez une procédure qui possède plusieurs arguments facultatifs. Si vous fournissez des arguments par nom, vous n'êtes pas obligé d'utiliser des virgules successives pour signaler l'absence d'arguments positionnels. Le fait de passer des arguments par nom facilite également le suivi des arguments qui sont passés ou de ceux qui sont omis.

Restrictions sur les spécifications d'arguments par nom

Vous ne pouvez pas passer d'arguments par nom pour éviter d'entrer des arguments requis. Vous ne pouvez omettre que les arguments facultatifs.

Vous ne pouvez pas passer un tableau de paramètres par nom. En effet, lorsque vous appelez la procédure, vous devez fournir un nombre indéfini d'arguments séparés par des virgules pour le tableau de paramètres, or, le compilateur ne peut pas associer plusieurs arguments à un seul nom.

Voir aussi

Tâches

Comment : passer des arguments à une procédure
Comment : passer des arguments à une procédure par nom

Référence

Optional (Visual Basic)
ParamArray

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Paramètres facultatifs
Tableaux de paramètres

Comment : passer des arguments à une procédure par nom

Lorsque vous appelez une procédure **Sub** ou **Function**, vous pouvez passer des arguments à celle-ci *par nom*, sans tenir compte de l'ordre dans lequel les paramètres correspondants apparaissent dans la définition de la procédure.

Le passage des arguments par nom facilite la lecture de votre code appelant parce qu'elle donne un sens aux valeurs d'argument dans l'appel de procédure. Si la procédure a des paramètres facultatifs, le passage par nom permet également de suivre facilement les arguments que vous passez et ceux que vous omettez.

Pour obtenir des règles et des restrictions sur la fourniture des arguments par nom, consultez Passage des arguments par position et par nom.

Pour passer un argument par nom

1. Examinez le code source de la déclaration de la procédure pour déterminer l'orthographe exacte des noms de paramètres.
2. Dans le code appelant, préparez l'appel de procédure normalement, avec la liste d'arguments entre parenthèses après le nom de procédure.
3. Pour chaque argument vous envisagez de passer par nom, fournissez le nom déclaré du paramètre suivi de deux-points et un signe égal (**:=**), suivis de la valeur que vous passez pour l'argument.
4. Vous pouvez fournir des arguments nommés dans n'importe quel ordre, mais vous devez fournir tous les arguments de position avant de fournir tout argument nommé.

Exemple

L'exemple suivant affiche une procédure **Sub** avec trois paramètres et un Call, instruction (Visual Basic) qui passe des arguments à ces paramètres par nom.

VB

```
Sub studentInfo(ByVal name As String, _
       Optional ByVal age As Short = 0, _
       Optional ByVal birth As Date = #1/1/2000#)

  Debug.WriteLine("Name = " & name & _
                  "; age = " & CStr(age) & _
                  "; birth date = " & CStr(birth))
End Sub
```

VB

```
Call studentInfo(age:=19, birth:=#9/21/1981#, name:="Mary")
```

Lorsque vous passez des arguments par nom, vous ne devez pas les fournir dans le même ordre que celui utilisé par la procédure pour les déclarer.

Compilation du code

Assurez-vous que le nom de paramètre dans la liste d'arguments correspond exactement au nom déclaré par la procédure.

Voir aussi

Tâches

Comment : passer des arguments à une procédure

Référence

Optional (Visual Basic)

ParamArray

Concepts

Procédures dans Visual Basic

Paramètres et arguments d'une procédure

Passage d'arguments par valeur et par référence

Passage des arguments par position et par nom
Paramètres facultatifs
Tableaux de paramètres

Paramètres facultatifs

Vous pouvez spécifier qu'un paramètre de procédure est facultatif et qu'il n'est pas nécessaire de fournir un argument lorsque la procédure est appelée. Les *arguments facultatifs* sont indiqués à l'aide du mot clé **Optional** dans la définition de la procédure. Les règles suivantes s'appliquent :

- Chaque paramètre facultatif dans la définition de la procédure doit spécifier une valeur par défaut.
- La valeur par défaut d'un paramètre facultatif doit être une expression constante.
- Tous les paramètres qui suivent un paramètre facultatif dans la définition de la procédure doivent également être facultatifs.

La syntaxe suivante montre une déclaration de procédure comprenant un paramètre facultatif :

```
Sub sub name(ByVal parameter 1 As data type 1, Optional ByVal parameter 2 As data type 2 =
default value)
```

Appel de procédures à l'aide de paramètres facultatifs

Lorsque vous appelez une procédure à l'aide d'un paramètre facultatif, vous pouvez choisir de fournir l'argument ou non. Dans le cas contraire, la procédure utilise la valeur par défaut déclarée pour ce paramètre.

Pour omettre un ou plusieurs arguments facultatifs dans la liste des arguments, utilisez des virgules successives afin de marquer leurs positions. L'exemple d'appel suivant fournit le premier et le quatrième argument, mais pas le deuxième ni le troisième :

```
Call sub name(argument 1, , , argument 4)
```

Détermination de la présence d'un argument facultatif

Une procédure ne peut pas détecter au moment de l'exécution si un argument donné a été omis ou si le code appelant a explicitement fourni la valeur par défaut. Pour établir cette distinction, vous pouvez définir une valeur improbable comme valeur par défaut. La procédure suivante définit le paramètre facultatif `office`, puis teste sa valeur par défaut, `QJZ`, pour déterminer s'il a été omis dans l'appel :

VB

```
Sub notify(ByVal company As String, Optional ByVal office As String = "QJZ")
    If office = "QJZ" Then
        Debug.WriteLine("office not supplied -- using Headquarters")
        office = "Headquarters"
    End If
    ' Insert code to notify headquarters or specified office.
End Sub
```

Si le paramètre facultatif est un type référence tel que **String**, vous pouvez utiliser **Nothing** comme valeur par défaut, à condition qu'il ne s'agisse pas d'une valeur attendue pour l'argument.

Paramètres facultatifs et surcharge

La surcharge est une autre méthode permettant de définir une procédure à l'aide de paramètres facultatifs. Si vous disposez d'un seul paramètre facultatif, vous pouvez définir deux versions surchargées de la procédure, l'une avec le paramètre et l'autre sans. Cette approche se complique au fur et à mesure que le nombre de paramètres facultatifs augmente. Cependant, vous avez l'avantage de savoir avec certitude si le programme appelant a fourni chaque argument facultatif.

Voir aussi

Tâches

Comment : définir des paramètres optionnels pour une procédure
Comment : appeler une procédure qui accepte des paramètres optionnels
Comment : déterminer si un paramètre optionnel a été fourni

Référence

Optional (Visual Basic)

ParamArray

Concepts

Procédures dans Visual Basic

Paramètres et arguments d'une procédure

Passage d'arguments par valeur et par référence

Passage des arguments par position et par nom

Tableaux de paramètres

Surcharge de procédure

Comment : définir des paramètres optionnels pour une procédure

Vous pouvez spécifier qu'un paramètre de procédure est facultatif, afin que le code appelant ne doive pas fournir un argument pour celle-ci lorsqu'il appelle la procédure. Lors de cette opération, vous définissez une valeur par défaut que la procédure utilise si l'argument n'est pas fourni.

Vous pouvez définir plusieurs paramètres facultatifs, mais tous les paramètres facultatifs doivent être à la fin de la liste de paramètres. Tous les paramètres requis doivent précéder chaque paramètre facultatif.

Pour définir un paramètre facultatif

1. Dans la déclaration de procédure, faites précéder le nom de paramètre dans la liste de paramètres du mot clé **Optional**.
2. Comme d'habitude, faites suivre le nom de paramètre d'une clause **As** et faites suivre la clause **As** d'un signe égal (**=**).
3. Faites suivre le signe égal de la valeur par défaut du paramètre. Celle-ci doit être une expression constante, afin que le compilateur puisse l'évaluer complètement au moment de la compilation.
4. Vous devez déclarer chaque paramètre suivant comme **Optional**.

Exemple

L'exemple suivant montre une déclaration de procédure comprenant un paramètre facultatif :

VB

```
Sub notify(ByVal company As String, Optional ByVal office As String = "QJZ")
    If office = "QJZ" Then
        Debug.WriteLine("office not supplied -- using Headquarters")
        office = "Headquarters"
    End If
    ' Insert code to notify headquarters or specified office.
End Sub
```

Si le code appelant ne fournit pas de valeur pour `office` dans la liste d'arguments, Visual Basic fournit la valeur par défaut "QJZ".

Compilation du code

Vous devez spécifier une valeur par défaut pour chaque paramètre optionnel dans la déclaration de procédure. Assurez-vous que chaque valeur par défaut est une constante que le compilateur peut évaluer au moment de la compilation.

Voir aussi

Tâches

Comment : appeler une procédure qui accepte des paramètres optionnels
Comment : déterminer si un paramètre optionnel a été fourni

Référence

Optional (Visual Basic)
ParamArray

Concepts

Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Passage des arguments par position et par nom
Paramètres facultatifs
Tableaux de paramètres
Surcharge de procédure

Comment : appeler une procédure qui accepte des paramètres optionnels

Lorsque vous appelez une procédure à l'aide d'un paramètre facultatif, vous pouvez choisir de fournir l'argument correspondant ou non. Dans le cas contraire, la procédure utilise la valeur par défaut déclarée pour ce paramètre.

- Si vous avez l'intention de fournir l'argument, incluez-le normalement dans la liste d'arguments, entre virgules.
- Si vous avez l'intention d'omettre l'argument, utilisez les virgules consécutives dans la liste d'arguments pour marquer son omission.
- Si vous omettez l'argument et vous fournissez des arguments par nom, vous ne devez pas indiquer l'argument omis que ce soit par nom ou par virgules.

Exemple

L'exemple suivant fait plusieurs appels à la fonction **MsgBox**. **MsgBox** a un paramètre requis et deux paramètres facultatifs.

VB

```
MsgBox("Important message", MsgBoxStyle.Critical, "MsgBox Example")
MsgBox("Just display this message.")
MsgBox("Test message", , "Title bar text")
MsgBox(Title:="Title bar text", Prompt:="Test message")
```

Le premier appel à **MsgBox** fournit les trois arguments dans l'ordre. **MsgBox** les définit. Le deuxième appel fournit uniquement l'argument requis. Le troisième et le quatrième appel fournissent le premier et le troisième argument. Le troisième appel le fait par position et le quatrième appel le fait par nom.

Compilation du code

Avant d'omettre un argument dans la liste d'arguments, assurez-vous que le paramètre correspondant est facultatif, assurez-vous également que vous souhaitez que la procédure utilise la valeur par défaut pour ce paramètre.

Si vous fournissez un argument par nom, assurez-vous que le nom dans la liste d'arguments correspond exactement au nom de paramètre déclaré.

Voir aussi

Tâches

Comment : définir des paramètres optionnels pour une procédure
Comment : déterminer si un paramètre optionnel a été fourni

Référence

Optional (Visual Basic)
ParamArray

Concepts

Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Passage des arguments par position et par nom
Paramètres facultatifs
Tableaux de paramètres
Surcharge de procédure

Comment : déterminer si un paramètre optionnel a été fourni

Lorsqu'une procédure définit un paramètre optionnel, il peut s'avérer nécessaire de déterminer si le code appelant a fourni ou omis l'argument correspondant.

Il y aurait deux raisons pour lesquelles la valeur de paramètre compare une valeur égale à sa valeur par défaut :

- Le code appelant a omis l'argument dans l'appel de procédure ; ou
- Le code appelant a fourni l'argument avec une valeur égale à la valeur par défaut du paramètre.

Le code de procédure ne peut pas faire la différence entre ces deux possibilités. Cela n'a généralement pas d'importance, mais dans certains cas la procédure doit prendre des mesures différentes pour chaque possibilité. La meilleure approche consiste à définir une valeur improbable comme valeur par défaut, bien que le code appelant puisse tout de même fournir cette valeur.

Si vous attachez une grande importance à ce que le programme appelant fournisse un argument facultatif, l'approche la plus sûre consiste à définir des versions surchargées de la procédure. Consultez
Comment : définir plusieurs versions d'une procédure et Considérations sur les surcharges de procédures.

Pour déterminer si un argument est passé à un paramètre optionnel

1. Définissez une valeur extrêmement improbable comme valeur par défaut pour le paramètre.
2. Si le paramètre facultatif est un type référence tel que **String**, vous pouvez utiliser **Nothing** comme valeur par défaut, à condition qu'il ne s'agisse pas d'une valeur attendue pour l'argument.
3. Dans le code de procédure, comparez le paramètre à la valeur par défaut et prenez la mesure appropriée.

Surcharge d'une procédure avec et sans paramètre facultatif

La surcharge est une autre manière de définir une procédure avec des paramètres facultatifs. Si vous disposez d'un seul paramètre facultatif, vous pouvez définir deux versions surchargées de la procédure, l'une acceptant le paramètre et l'autre pas. Cette approche se complique au fur et à mesure que le nombre de paramètres facultatifs augmente. Cependant, vous avez l'avantage de savoir avec certitude si le programme appelant a fourni chaque argument facultatif.

Pour définir plusieurs versions de la procédure pour couvrir l'inclusion et omission d'un argument

1. Définissez une version de la procédure avec le paramètre dans la liste d'arguments. Ne déclarez pas le paramètre comme **Optional**.
2. Définissez une autre version de la procédure sans le paramètre. La déclaration doit être identique à celle de la première version dans tous les autres cas.
3. Placez le code approprié à chaque appel dans la version respective de la procédure.

Exemple

La procédure suivante définit le paramètre facultatif `office`, puis teste sa valeur par défaut, `QJZ`, pour déterminer s'il a été omis dans l'appel.

VB

```
Sub notify(ByVal company As String, Optional ByVal office As String = "QJZ")
    If office = "QJZ" Then
        Debug.WriteLine("office not supplied -- using Headquarters")
        office = "Headquarters"
    End If
    ' Insert code to notify headquarters or specified office.
End Sub
```

Si le code appelant ne fournit pas de valeur pour `office` dans la liste d'arguments, Visual Basic fournit la valeur par défaut "QJZ".

Si le paramètre facultatif est un type référence tel que **String**, vous pouvez utiliser Nothing (Visual Basic) comme valeur par défaut, à condition qu'il ne s'agisse pas d'une valeur attendue pour l'argument.

Pour obtenir un exemple d'utilisation de surcharges pour déterminer si un paramètre optionnel est été passé, consultez Comment : surcharger une procédure qui accepte des paramètres optionnels.

Voir aussi

Tâches

Comment : définir des paramètres optionnels pour une procédure
Comment : appeler une procédure qui accepte des paramètres optionnels

Référence

Optional (Visual Basic)
ParamArray

Concepts

Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Passage des arguments par position et par nom
Paramètres facultatifs
Tableaux de paramètres
Surcharge de procédure

Tableaux de paramètres

En règle générale, vous ne pouvez pas appeler une procédure avec un nombre d'arguments supérieur à celui de la déclaration de procédure. Lorsque vous avez besoin d'un nombre indéfini d'arguments, vous pouvez déclarer un *tableau de paramètres*, lequel permet à la procédure d'accepter un tableau de valeurs comme paramètre. Il n'est pas nécessaire de connaître le nombre d'éléments contenus dans le tableau de paramètres lorsque vous définissez la procédure. La taille du tableau est déterminée individuellement par chaque appel à la procédure.

Déclaration d'un ParamArray

Vous devez utiliser le mot clé ParamArray pour désigner un tableau de paramètres dans la liste des paramètres. Les règles suivantes s'appliquent :

- Une procédure ne peut définir qu'un tableau de paramètres, qui doit représenter le dernier paramètre dans la définition de la procédure.
- Le tableau de paramètres doit être passé par valeur. En programmation, il est conseillé d'inclure explicitement le mot clé ByVal dans la définition de la procédure.
- Le tableau de paramètres est automatiquement facultatif. Sa valeur par défaut est un tableau unidimensionnel vide dont le type d'élément est tableau de paramètres.
- Tous les paramètres précédant le tableau de paramètres sont requis. Le tableau de paramètres doit être le seul paramètre facultatif.

Pour plus d'informations, consultez Comment : définir une procédure avec un nombre indéfini de paramètres.

Appel d'un ParamArray

Lorsque vous appelez une procédure qui définit un tableau de paramètres, vous pouvez fournir l'argument de l'une des manières suivantes:

- Aucune valeur, ce qui signifie que vous pouvez omettre l'argument ParamArray. Dans ce cas, un tableau vide est passé à la procédure. Vous pouvez également passer le mot clé Nothing (Visual Basic), qui produit le même effet.
- Liste d'un nombre arbitraire d'arguments séparés par des virgules. Le type de données de chaque argument doit être implicitement convertible en type d'élément **ParamArray**.
- Tableau avec le même type d'élément que celui du tableau de paramètres.

Dans tous les cas, le code de la procédure traite le tableau de paramètres comme un tableau unidimensionnel dont chaque élément est du même type de données que le type de données **ParamArray**.

Pour plus d'informations, consultez Comment : appeler une procédure qui accepte un nombre indéfini de paramètres.

Remarque de sécurité
Si vous travaillez dans un tableau dont la taille peut être indéfinie, vous risquez de saturer la capacité interne de votre application. Si vous acceptez un tableau de paramètres, vous devez tester la taille du tableau auquel le code appelant est passé. Effectuez des étapes appropriées si le tableau est trop grand pour votre application. Pour plus d'informations, consultez Comment : déterminer la taille d'un tableau.

Voir aussi

Référence

Optional (Visual Basic)
UBound, fonction (Visual Basic)

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Passage des arguments par position et par nom
Paramètres facultatifs
Surcharge de procédure

Vérification du type dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : définir une procédure avec un nombre indéfini de paramètres

Vous pouvez déclarer un *tableau de paramètres* comme dernière entrée dans la liste de paramètres d'une procédure. Cela permet à la procédure d'accepter un jeu de valeurs pour ce paramètre, au lieu d'une seule valeur. Vous n'êtes pas obligé de connaître le nombre de valeurs lors de la définition de la procédure. Le jeu est déterminé individuellement par chaque appel à la procédure, et chaque appel peut passer un nombre différent de valeurs.

Pour plus d'informations, consultez Tableaux de paramètres.

Pour définir une procédure qui peut accepter un nombre indéfini de valeurs pour son dernier paramètre

1. Dans la déclaration de la procédure, définissez normalement la liste de paramètres. Tous les paramètres excepté le dernier doivent être requis (et non Optional (Visual Basic)).
2. Précédez le dernier nom de paramètre avec les mots clé **ByVal ParamArray**. Ce paramètre est automatiquement facultatif. N'incluez pas le mot clé **Optional**.
3. Faites suivre le nom de tableau de paramètres par des parenthèses vides.
4. Faites suivre les parenthèses vides de la clause **As** habituelle.
5. Ne faites pas suivre la clause **As** d'une valeur par défaut. La valeur par défaut du tableau de paramètres représente automatiquement un tableau unidimensionnel vide du type de données que vous avez spécifié dans la clause **As**.

Utilisation des valeurs de tableau de paramètres

Le code de la procédure doit traiter le tableau de paramètres comme un tableau unidimensionnel dont chaque élément est du même type de données que le type de données **ParamArray**.

Pour accéder à l'une des valeurs d'un tableau de paramètres

1. Dans le code de procédure, déterminez la longueur du tableau passée au tableau de paramètres en appelant le UBound, fonction (Visual Basic) sur le nom de tableau de paramètres.
2. Dans une instruction exécutable dans le code de procédure, faites suivre le nom de tableau de paramètres par un indice entre parenthèses. Cet indice doit être compris entre 0 et la limite supérieure retournées par **UBound**.

Remarque de sécurité
Si vous travaillez dans un tableau dont la taille peut être indéfinie, vous risquez de saturer la capacité interne de votre applica tion. Si vous acceptez un tableau de paramètres à partir du code appelant, vous devez tester sa longueur et prendre des mes ures appropriées s'il est trop grand pour votre application.

Exemple

L'exemple suivant définit une procédure avec un tableau de paramètres, et renvoie les valeurs de tous les éléments de tableau passés au tableau de paramètres.

VB

```
Sub studentScores(ByVal name As String, ByVal ParamArray scores() As String)
    Debug.WriteLine("Scores for " & name & ":" & vbCrLf)
    ' Use UBound to determine largest subscript of the array.
    For i As Integer = 0 To UBound(scores, 1)
        Debug.WriteLine("Score " & i & ": " & scores(i))
    Next i
End Sub
```

Les exemples suivants montrent des appels classiques à `studentScores`.

VB

```
Call studentScores("Anne", "10", "26", "32", "15", "22", "24", "16")
Call studentScores("Mary", "High", "Low", "Average", "High")
Dim JohnScores() As String = {"35", "Absent", "21", "30"}
Call studentScores("John", JohnScores)
```

Compilation du code

Assurez-vous que le **ParamArray** est le dernier paramètre dans la liste de paramètres, et qu'aucun des paramètres précédents n'est déclaré **Optional**.

Voir aussi

Tâches

Comment : appeler une procédure qui accepte un nombre indéfini de paramètres

Référence

ByVal

ParamArray

Concepts

Paramètres et arguments d'une procédure

Passage d'arguments par valeur et par référence

Passage des arguments par position et par nom

Paramètres facultatifs

Surcharge de procédure

Vérification du type dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Comment : appeler une procédure qui accepte un nombre indéfini de paramètres

Une procédure peut déclarer la dernière entrée dans sa liste de paramètres qui doit être un *tableau de paramètres*. Cela lui permet d'accepter un nombre indéfini de valeurs pour ce paramètre au lieu d'une seule valeur.

Pour plus d'informations, consultez Tableaux de paramètres.

Pour appeler une procédure avec un tableau de paramètres et omettre l'argument correspondant

1. Écrivez normalement l'appel de procédure. Le tableau de paramètres doit être le dernier argument.
2. Terminez la liste d'arguments avec l'avant-dernier argument. Le tableau de paramètres est facultatif et tous les paramètres précédents doivent être requis.

   - ou -

   Fournissez le mot clé **Nothing** comme argument pour le tableau de paramètres.
3. Visual Basic passe un tableau unidimensionnel vide à la procédure pour le tableau de paramètres.

Pour appeler une procédure avec un tableau de paramètres et fournir une liste d'arguments

1. Écrivez normalement l'appel de procédure. Le tableau de paramètres doit être le dernier argument.
2. Fournissez un nombre d'arguments, séparé par des virgules, pour le tableau de paramètres. Le type de données de chaque argument doit être implicitement convertible en type d'élément **ParamArray**.
3. Visual Basic passe un tableau unidimensionnel à la procédure, contenant toutes les valeurs que vous avez fournies.

Pour appeler une procédure avec un tableau de paramètres et fournir un tableau d'arguments

1. Écrivez normalement l'appel de procédure. Le tableau de paramètres doit être le dernier argument.
2. Pour le tableau de paramètres, fournissez un tableau unidimensionnel avec le même type d'élément que le type d'élément du tableau de paramètres.
3. Visual Basic passe votre tableau à la procédure.

Exemple

Les exemples suivants illustrent des appels classiques à la procédure `studentScores` définis dans Comment : définir une procédure avec un nombre indéfini de paramètres.

VB

```
    Call studentScores("George")
...    Call studentScores("Anne", "10", "26", "32", "15", "22", "24", "16")
    Call studentScores("Mary", "High", "Low", "Average", "High")
    Dim JohnScores() As String = {"35", "Absent", "21", "30"}
    Call studentScores("John", JohnScores)
```

Le premier appel omet complètement le tableau de paramètres et fournit uniquement le premier argument requis. La procédure `studentScores` traite cet appel comme le passage d'un tableau vide.

Les deuxième et troisième appels fournissent des listes d'arguments de longueurs différentes au tableau de paramètres. Chacune de ces listes est passée comme un tableau de valeurs.

Le quatrième appel passe un tableau au tableau de paramètres.

Voir aussi

Référence

Optional (Visual Basic)
ParamArray

ByVal
UBound, fonction (Visual Basic)

Concepts

Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Passage des arguments par position et par nom
Paramètres facultatifs
Surcharge de procédure
Vérification du type dans Visual Basic

Procédures récursives

Une procédure *récursive* est une procédure qui s'appelle elle-même. En général, il ne s'agit pas de la façon la plus efficace pour écrire du code Visual Basic.

La procédure suivante utilise la récurrence pour calculer la factorielle de son argument d'origine.

VB

```
Function factorial(ByVal n As Integer) As Integer
    If n <= 1 Then
        Return 1
    Else
        Return factorial(n - 1) * n
    End If
End Function
```

Considérations sur les procédures récursives

Conditions de limitation Vous devez concevoir une procédure récursive à tester pour au moins une condition qui peut mettre fin à la récurrence, et vous devez également gérer les situations pour lesquelles aucune de ces conditions n'est satisfaite pour un nombre raisonnable d'appels récursifs. Si aucune condition ne peut être satisfaite sans erreur, votre procédure s'expose à un risque élevé d'exécution dans une boucle infinie.

Utilisation de la mémoire. Votre application dispose d'un espace limité pour les variables locales. Chaque fois qu'une procédure s'appelle elle-même, elle utilise davantage d'espace pour les copies supplémentaires de ses variables locales. Si le processus se poursuit indéfiniment, il peut se produire une erreur StackOverflowException.

Efficacité. Vous pouvez presque toujours remplacer la récurrence par une boucle. Une boucle ne possède pas la charge mémoire pour passer des arguments, initialiser le stockage supplémentaire et retourner des valeurs. Votre performance peut être considérablement améliorée sans appels récurrents.

Récurrence mutuelle Vous pouvez observer une performance médiocre ou même une boucle infinie si deux procédures s'appellent l'une et l'autre. Ce type de conception présente les mêmes problèmes qu'une procédure récursive simple, mais peut être plus difficile à détecter et à déboguer.

Appel avec parenthèses. Lorsqu'une procédure **Function** s'appelle de manière récursive, le nom de cette procédure doit être suivi de parenthèses, même s'il n'y a pas de liste d'arguments. Sinon, le nom de la fonction est considéré comme sa valeur de retour.

Test Si vous écrivez une procédure récursive, vous devez la tester soigneusement afin de vous assurer qu'elle satisfait toujours certaines conditions de limitation. Vous devez également veiller à toujours disposer d'une mémoire suffisante en dépit du grand nombre d'appels récurrents.

Voir aussi

Tâches

Procédures de dépannage
Dépannage des exceptions : System.StackOverflowException

Référence

StackOverflowException

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure
Surcharge de procédure
Structures de boucle

Surcharge de procédure

La *surcharge* d'une procédure consiste à définir celle-ci en plusieurs versions, à l'aide du même nom, mais avec des listes de paramètres différentes. La finalité d'une surcharge est de définir plusieurs versions de procédures étroitement liées sans avoir à les différencier par nom. Pour ce faire, vous devez faire varier la liste des paramètres.

Règles de surcharge

Lorsque vous surchargez une procédure, les règles suivantes s'appliquent :

- **Même nom**. Chaque version surchargée doit utiliser le même nom de procédure.
- **Signature différente**. Chaque version surchargée doit être différente de toutes les autres versions surchargées sur au moins l'un des points suivants :
    - Nombre de paramètres
    - Ordre des paramètres
    - Types de données des paramètres
    - Nombre de paramètres de type (pour une procédure générique)
    - Type de retour (uniquement pour un opérateur de conversion)

  Avec le nom de procédure, les éléments précédents sont appelés collectivement *signature* de la procédure. Lorsque vous appelez une procédure surchargée, le compilateur utilise la signature pour vérifier que l'appel correspond bien à la définition.
- **Éléments ne faisant pas partie de la signature**. Vous ne pouvez pas surcharger une procédure sans faire varier la signature. En particulier, vous ne pouvez pas surcharger une procédure en faisant varier uniquement un ou plusieurs des éléments suivants :
    - Mots clés de modificateur de procédure, tels que **Public**, **Shared** et **Static**
    - Noms de paramètres ou de paramètres de type
    - Contraintes de paramètre de type (pour une procédure générique)
    - Mots clés de modificateur de paramètre, tels que **ByRef** et **Optional**
    - S'il retourne une valeur
    - Type de données de la valeur de retour (à l'exception d'un opérateur de conversion)

  Les éléments de la liste précédente ne font pas partie de la signature. Bien que vous ne puissiez pas les utiliser pour différencier des versions surchargées, vous pouvez les faire varier parmi des versions surchargées qui sont correctement différenciées par leurs signatures.
- **Arguments à liaison tardive**. Si vous essayez de passer une variable objet à liaison tardive à une procédure surchargée, vous devez déclarer le paramètre approprié comme Object.

Versions multiples d'une procédure

Supposons que vous écriviez une procédure **Sub** pour publier une transaction relative au solde d'un client, et que vous vouliez faire référence au client par son nom ou son numéro de compte. Pour ce faire, vous pouvez définir deux procédures **Sub** différentes, comme dans l'exemple suivant :

VB

```
Sub postName(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
Sub postAcct(ByVal custAcct As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

Versions surchargées

La première solution consiste à surcharger un seul nom de procédure. Vous pouvez utiliser le mot clé Overloads pour définir une version de la procédure pour chaque liste de paramètres comme suit :

VB

```
Overloads Sub post(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
Overloads Sub post(ByVal custAcct As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

Surcharges supplémentaires

Si vous voulez également accepter le montant d'une transaction sous forme d'un type **Decimal** ou **Single**, vous pouvez surcharger davantage `post` pour permettre cette variation. Si vous avez effectué cette opération sur chacune des surcharges de l'exemple précédent, vous devez obtenir quatre procédures **Sub** portant toutes le même nom, mais avec quatre signatures différentes.

Avantages de la surcharge

L'avantage présenté par la surcharge d'une procédure réside dans la flexibilité de l'appel. Pour utiliser la procédure `post` déclarée dans l'exemple précédent, le code appelant peut obtenir l'identification du client selon un type **String** ou **Integer**, puis appeler la même procédure dans les deux cas. L'exemple suivant illustre ce comportement :

VB

```
Imports MSVB = Microsoft.VisualBasic
```

VB

```
Dim customer As String
Dim accountNum As Integer
Dim amount As Single
customer = MSVB.Interaction.InputBox("Enter customer name or number")
amount = MSVB.Interaction.InputBox("Enter transaction amount")
Try
    accountNum = CInt(customer)
    Call post(accountNum, amount)
Catch
    Call post(customer, amount)
End Try
```

Voir aussi

Tâches

Comment : définir plusieurs versions d'une procédure
Comment : appeler une procédure surchargée
Comment : surcharger une procédure qui accepte des paramètres optionnels
Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Référence

Overloads

Concepts

Procédures dans Visual Basic
Considérations sur les surcharges de procédures
Résolution de surcharge
Types génériques en Visual Basic

Comment : définir plusieurs versions d'une procédure

Vous pouvez définir une procédure dans plusieurs versions en la *surchargeant*, à l'aide du même nom, mais avec une liste de paramètres différente pour chaque version. La finalité d'une surcharge est de définir plusieurs versions de procédures étroitement liées sans avoir à les différencier par nom.

Pour plus d'informations, consultez Surcharge de procédure.

Pour définir plusieurs versions d'une procédure

1. Écrivez une instruction de déclaration **Sub** ou **Function** pour chaque version de la procédure que vous souhaitez définir. Utilisez le même nom de procédure dans chaque déclaration.
2. Précédez le mot clé **Sub** ou **Function** dans chaque déclaration du mot clé Overloads. Vous pouvez omettre **Overloads** dans les déclarations, mais si vous l'incluez dans une des déclarations, vous devez l'inclure dans toutes les déclarations.
3. À la suite de chaque instruction de déclaration, écrivez le code de procédure pour gérer le cas spécifique où le code appelant fournit des arguments qui correspondent à la liste de paramètres de cette version. Vous ne devez pas tester pour quels paramètres le code appelant a fourni les paramètres. Visual Basic passe le contrôle à la version correspondante de votre procédure.
4. Terminez chaque version de la procédure avec l'instruction **End Sub** ou **End Function** selon le cas.

Exemple

L'exemple suivant définit une procédure **Sub** pour publier une transaction par rapport au compte d'un client. Il utilise le mot clé **Overloads** pour définir deux versions de la procédure, un qui accepte le client par son nom et l'autre par son numéro de compte.

VB

```
Overloads Sub post(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
Overloads Sub post(ByVal custAcct As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

Le code appelant peut obtenir l'identification de client en tant que **String** ou en tant que **Integer**, puis utiliser la même instruction appelante dans les deux cas.

Pour plus d'informations sur l'appel de ces versions de la procédure `post`, consultez Comment : appeler une procédure surchargée.

Compilation du code

Assurez-vous que chacune de vos versions surchargées a le même nom de procédure, mais une liste de paramètres différente.

Voir aussi

Tâches

Procédures de dépannage
Comment : surcharger une procédure qui accepte des paramètres optionnels
Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Considérations sur les surcharges de procédures
Résolution de surcharge

Comment : appeler une procédure surchargée

L'avantage présenté par la surcharge d'une procédure réside dans la flexibilité de l'appel. Le code appelant peut obtenir les informations dont il a besoin pour passer à la procédure, puis appeler un nom de procédure simple, quels que soient les arguments qu'il passe.

Pour appeler une procédure qui possède plusieurs versions définies

1. Dans le code appelant, déterminez les données à passer à la procédure.
2. Écrivez l'appel de procédure normalement, en présentant les données dans la liste d'arguments. Assurez-vous que les arguments correspondent à la liste de paramètres dans l'une des versions définies pour la procédure.
3. Vous ne devez pas déterminer la version de la procédure à appeler. Visual Basic passe le contrôle à la version correspondant à votre liste d'arguments.

   L'exemple suivant appelle la procédure `post` déclarée dans Comment : définir plusieurs versions d'une procédure. Il obtient l'identification de client, détermine si c'est un **String** ou un **Integer** puis, dans les deux cas, appelle la même procédure.

   VB

   ```
   Imports MSVB = Microsoft.VisualBasic
   ```

   VB

   ```
   Dim customer As String
   Dim accountNum As Integer
   Dim amount As Single
   customer = MSVB.Interaction.InputBox("Enter customer name or number")
   amount = MSVB.Interaction.InputBox("Enter transaction amount")
   Try
       accountNum = CInt(customer)
       Call post(accountNum, amount)
   Catch
       Call post(customer, amount)
   End Try
   ```

Voir aussi

Tâches

Procédures de dépannage
Comment : définir plusieurs versions d'une procédure
Comment : surcharger une procédure qui accepte des paramètres optionnels
Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Référence

Overloads

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Surcharge de procédure
Considérations sur les surcharges de procédures
Résolution de surcharge

Comment : surcharger une procédure qui accepte des paramètres optionnels

Si une procédure possède un ou plusieurs paramètres Optional (Visual Basic), vous ne pouvez pas définir de version surchargée correspondant une de ses surcharges implicites. Pour plus d'informations, consultez des "Implicit Overloads for Optional Parameters" dans Considérations sur les surcharges de procédures.

Paramètre facultatif unique

Pour surcharger une procédure qui accepte un paramètre facultatif

1. Écrivez une instruction de déclaration **Sub** ou **Function** qui inclut le paramètre facultatif dans la liste de paramètres. N'utilisez pas le mot clé **Optional** dans cette version surchargée.
2. Faites précéder le mot clé **Sub** ou **Function** du mot clé Overloads.
3. Écrivez le code de procédure qui doit être exécuté lorsque le code appelant fournit l'argument facultatif.
4. Terminez la procédure par l'instruction **End Sub** ou **End Function**, selon le cas.
5. Écrivez une deuxième instruction de déclaration qui est identique à la première déclaration, excepté qu'elle n'inclut pas le paramètre facultatif dans la liste de paramètres.
6. Écrivez le code de procédure qui doit être exécuté lorsque le code appelant ne fournit pas l'argument facultatif. Terminez la procédure par l'instruction **End Sub** ou **End Function**, selon le cas.

   L'exemple suivant affiche une procédure définie par un paramètre facultatif, un jeu d'équivalents de deux procédures surchargées et enfin des exemples de versions surchargées non valides et valides.

   VB

```
Sub q(ByVal b As Byte, Optional ByVal j As Long = 6)
```

   VB

```
' The preceding definition is equivalent to the following two overloads.
' Overloads Sub q(ByVal b As Byte)
' Overloads Sub q(ByVal b As Byte, ByVal j As Long)
```

   VB

```
' Therefore, the following overload is not valid because the signature is already in u
se.
' Overloads Sub q(ByVal c As Byte, ByVal k As Long)
' The following overload uses a different signature and is valid.
Overloads Sub q(ByVal b As Byte, ByVal j As Long, ByVal s As Single)
```

Paramètres facultatifs multiples

Pour créer une procédure disposant de plusieurs paramètres facultatifs, vous avez normalement besoin de plus de deux versions surchargées. Par exemple, s'il y a deux paramètres optionnels, et le code appelant peut fournir ou omettre indépendamment chacun de l'autre, vous avez besoin de quatre versions surchargées, une pour chaque combinaison possible d'arguments fournis.

Comme le nombre de paramètres facultatifs augmente, la complexité de la surcharge augmente également. À moins que certaines combinaisons d'arguments fournies ne soient pas acceptables, pour *N* paramètres facultatifs, vous devez disposer de *2 ^ N* versions surchargées. Selon la nature de la procédure, vous pouvez trouver que la clarté de logique justifie l'effort supplémentaire de définition de toutes les versions surchargées.

Pour surcharger une procédure qui accepte plusieurs paramètres facultatifs

1. Déterminez quelles combinaisons d'arguments facultatifs fournis sont acceptables par rapport à la logique de la procédure. Une combinaison inacceptable peut survenir si un paramètre facultatif dépend d'un autre. Par exemple, si un paramètre accepte le nom du conjoint et qu'un autre accepte son âge, une combinaison d'arguments fournissant l'âge, mais omettant le nom est inacceptable.

2. Pour chaque combinaison acceptable d'arguments facultatifs fournis, écrivez une instruction de déclaration **Sub** ou **Function** qui définit la liste de paramètres correspondante. N'utilisez pas le mot clé **Optional**.

3. Dans chaque déclaration, précédez le mot clé **Sub** ou **Function** par le mot clé Overloads.

4. Après chaque déclaration, écrivez le code de procédure à exécuter lorsque le code appelant fournit une liste d'arguments correspondant à la liste de paramètres de cette déclaration.

5. Terminez chaque procédure par l'instruction **End Sub** ou **End Function**, selon le cas.

Voir aussi

Tâches

Procédures de dépannage
Comment : définir plusieurs versions d'une procédure
Comment : appeler une procédure surchargée
Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Paramètres facultatifs
Tableaux de paramètres
Surcharge de procédure
Résolution de surcharge

Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Si une procédure dispose d'un paramètre ParamArray, vous ne pouvez pas définir une version surchargée acceptant un tableau unidimensionnel comme tableau de paramètres. Pour plus d'informations, consultez des "Implicit Overloads for a ParamArray Parameter" dans Considérations sur les surcharges de procédures.

Pour surcharger une procédure qui accepte un nombre variable de paramètres

1. Assurez-vous que la procédure et la logique du code appelant bénéficient davantage de versions surchargées que d'un paramètre **ParamArray**. Consultez des "Overloads and ParamArrays" dans Considérations sur les surcharges de procédures.
2. Déterminez le nombre de valeurs fournies que la procédure doit accepter dans la partie variable de la liste de paramètres. Elle peut n'accepter aucune valeur, ou accepter un tableau unidimensionnel unique.
3. Pour chaque nombre acceptable de valeurs fournies, écrivez une instruction de déclaration **Sub** ou **Function** qui définit la liste de paramètres correspondante. N'utilisez ni le mot clé **Optional** ni **ParamArray** dans la version surchargée.
4. Dans chaque déclaration, précédez le mot clé **Sub** ou **Function** par le mot clé Overloads.
5. Après chaque déclaration, écrivez le code de procédure à exécuter lorsque le code appelant fournit des valeurs correspondant à la liste de paramètres de cette déclaration.
6. Terminez chaque procédure par l'instruction **End Sub** ou **End Function**, selon le cas.

Exemple

L'exemple suivant affiche une procédure définie avec un paramètre ParamArray, puis un jeu équivalent de procédures surchargées.

VB

```
Sub p(ByVal d As Date, ByVal ParamArray c() As Char)
```

VB

```
' The preceding definition is equivalent to the following overloads.
' Overloads Sub p(ByVal d As Date)
' Overloads Sub p(ByVal d As Date, ByVal c() As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char, ByVal c2 As Char)
' And so on, with an additional Char argument in each successive overload.
```

Vous ne pouvez pas surcharger une procédure avec une liste de paramètres qui accepte un tableau unidimensionnel comme tableau de paramètres. Toutefois, vous pouvez utiliser les signatures des autres surcharges implicites. Les déclarations suivantes illustrent ce principe.

VB

```
' The following overload is not valid because it takes an array for the parameter array.
' Overloads Sub p(ByVal x As Date, ByVal y() As Char)
' The following overload takes a single value for the parameter array and is valid.
Overloads Sub p(ByVal z As Date, ByVal w As Char)
```

Le code des versions surchargées ne doit pas tester si le code appelant a fourni des valeurs pour le paramètre **ParamArray**, ou dans le cas échéant, doit indiquer le nombre. Visual Basic passe le contrôle à la version correspondant à la liste d'arguments appelante.

Compilation du code

Comme une procédure avec un paramètre **ParamArray** est équivalente à un jeu de versions surchargées, vous ne pouvez pas la surcharger avec une liste de paramètres correspondant à chacune de ces surcharges implicites. Pour plus d'informations, consultez Considérations sur les surcharges de procédures.

Sécurité

Si vous travaillez dans un tableau dont la taille peut être indéfinie, vous risquez de saturer la capacité interne de votre application. Si vous acceptez un tableau de paramètres, vous devez tester la longueur du tableau passée par le code appelant et effectuer les étapes appropriées s'il est trop long pour votre application.

Voir aussi

Tâches

Procédures de dépannage
Comment : définir plusieurs versions d'une procédure
Comment : appeler une procédure surchargée
Comment : surcharger une procédure qui accepte des paramètres optionnels

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Paramètres facultatifs
Tableaux de paramètres
Surcharge de procédure
Résolution de surcharge

Considérations sur les surcharges de procédures

Lorsque vous surchargez une procédure, vous devez utiliser une *signature* différente pour chaque version surchargée. Cela signifie généralement que chaque version doit spécifier une liste de paramètres différente. Pour plus d'informations, consultez « Signature différente » dans Surcharge de procédure.

Vous pouvez surcharger une procédure **Function** avec une procédure **Sub**, et inversement, à condition que leurs signatures soient différentes. Deux surcharges ne peuvent pas différer uniquement suivant que l'un a une valeur de retour et l'autre pas.

Vous pouvez surcharger une propriété de la même manière que vous surchargez une procédure et avec les mêmes restrictions. Toutefois, vous ne pouvez pas surcharger une procédure avec une propriété, ou inversement.

Alternatives aux versions surchargées

Vous disposez parfois d'alternatives aux versions surchargées, en particulier lorsque la présence d'arguments est facultative ou que leur nombre est variable.

N'oubliez pas que les arguments facultatifs ne sont pas nécessairement pris en charge par tous les langages et les tableaux de paramètres sont limités à Visual Basic. Si vous écrivez une procédure qui sera vraisemblablement appelée à partir du code écrit dans un des différents langages, les versions surchargées offrent la plus grande souplesse.

Surcharges et arguments facultatifs

Lorsque le code appelant peut éventuellement fournir ou omettre un ou plusieurs arguments, vous pouvez définir plusieurs versions surchargées ou utiliser des paramètres facultatifs.

Quand utiliser des versions surchargées

Pensez à définir une série de versions surchargées dans les cas suivants :

- La logique dans le code de procédure est très différente suivant que le code appelant fournit un argument facultatif ou pas.
- Le code de procédure ne peut pas vérifier de manière fiable si le code appelant a fourni un argument facultatif. Par exemple, c'est le cas s'il n'y a aucun candidat possible pour une valeur par défaut que le code appelant n'était pas supposé fournir.

Quand utiliser les paramètres facultatifs

Il peut être préférable d'utiliser un ou plusieurs des paramètres facultatifs dans les cas suivants :

- La seule action requise lorsque le code appelant ne fournit pas un argument facultatif est de définir le paramètre à une valeur par défaut. Dans ce cas, le code de procédure peut être moins compliqué si vous définissez une version unique avec un ou plusieurs paramètres **Optional**.

Pour plus d'informations, consultez Paramètres facultatifs.

Surcharges et ParamArray

Lorsque le code appelant peut passer un nombre variable d'arguments, vous pouvez définir plusieurs versions surchargées ou utiliser un tableau de paramètres.

Quand utiliser des versions surchargées

Pensez à définir une série de versions surchargées dans les cas suivants :

- Vous savez que le code appelant ne passe jamais plus qu'un petit nombre de valeurs au tableau de paramètres.
- La logique dans le code de procédure est très différente selon le nombre de valeurs passées par le code appelant.
- Le code appelant peut passer des valeurs de types de données différents.

Quand utiliser un tableau de paramètres

Un paramètre **ParamArray** est plus utile dans les cas suivants :

- Vous n'êtes pas en mesure de prévoir combien de valeurs le code appelant peut passer au tableau de paramètres et elles pourraient représenter un grand nombre.

- La logique de procédure implique de parcourir toutes les valeurs que le code appelant passe, en exécutant essentiellement les mêmes opérations sur chaque valeur.

Pour plus d'informations, consultez Tableaux de paramètres.

Surcharges implicites des paramètres facultatifs

Une procédure avec un paramètre Optional (Visual Basic) équivaut à deux procédures surchargées, l'une avec l'argument facultatif et l'autre sans cet argument. Vous ne pouvez pas surcharger une procédure avec une liste de paramètres correspondant à l'un ou l'autre de ceux-ci. Les déclarations suivantes illustrent ce principe.

VB

```
Overloads Sub q(ByVal b As Byte, Optional ByVal j As Long = 6)
```

VB

```
' The preceding definition is equivalent to the following two overloads.
' Overloads Sub q(ByVal b As Byte)
' Overloads Sub q(ByVal b As Byte, ByVal j As Long)
```

VB

```
' Therefore, the following overload is not valid because the signature is already in use.
' Overloads Sub q(ByVal c As Byte, ByVal k As Long)
' The following overload uses a different signature and is valid.
Overloads Sub q(ByVal b As Byte, ByVal j As Long, ByVal s As Single)
```

Dans le cas d'une procédure possédant plusieurs arguments facultatifs, il existe un ensemble de surcharges implicites, qui peuvent se produire de manière logique, comme dans l'exemple précédent.

Surcharges implicites pour un paramètre ParamArray

Le compilateur considère qu'une procédure comportant un paramètre ParamArray possède un nombre infini de surcharges, qui se différencient selon ce que le code appelant passe au tableau de paramètres, notamment :

- Une surcharge lorsque le code appelant ne fournit pas d'argument au **ParamArray**
- Une surcharge lorsque le code appelant fournit un tableau unidimensionnel du type d'élément **ParamArray**
- Pour chaque entier positif, une surcharge lorsque le code appelant fournit le nombre correspondant d'arguments dont le type d'élément est **ParamArray**

Les déclarations suivantes illustrent ces surcharges implicites.

VB

```
Overloads Sub p(ByVal d As Date, ByVal ParamArray c() As Char)
```

VB

```
' The preceding definition is equivalent to the following overloads.
' Overloads Sub p(ByVal d As Date)
' Overloads Sub p(ByVal d As Date, ByVal c() As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char, ByVal c2 As Char)
' And so on, with an additional Char argument in each successive overload.
```

Vous ne pouvez pas surcharger une procédure avec une liste de paramètres qui prend un tableau unidimensionnel pour le tableau de paramètres. Toutefois, vous pouvez utiliser les signatures des autres surcharges implicites. Les déclarations suivantes illustrent ce principe.

VB

```
' The following overload is not valid because it takes an array for the parameter array.
' Overloads Sub p(ByVal x As Date, ByVal y() As Char)
' The following overload takes a single value for the parameter array and is valid.
Overloads Sub p(ByVal z As Date, ByVal w As Char)
```

Programmation sans type en remplacement de la surcharge

Si vous souhaitez permettre au code appelant de passer des types de données différents à un paramètre, Programmation sans type dans Visual Basic est une approche alternative. Vous pouvez affecter la valeur **Off** au commutateur de vérification de type avec l'instruction Option Strict ou l'option du compilateur /optionstrict. Vous ne devez donc pas déclarer le type de données du paramètre. Cependant, cette approche présente les inconvénients suivants par rapport à la surcharge :

- La programmation sans type génère une exécution moins efficace du code.
- La procédure doit tester tous les types de données pouvant être passés.
- Le compilateur ne peut pas signaler d'erreur si le code appelant passe un type de données non pris en charge par la procédure.

Voir aussi

Tâches

Procédures de dépannage
Comment : définir plusieurs versions d'une procédure
Comment : appeler une procédure surchargée
Comment : surcharger une procédure qui accepte des paramètres optionnels
Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Référence

Overloads

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Résolution de surcharge

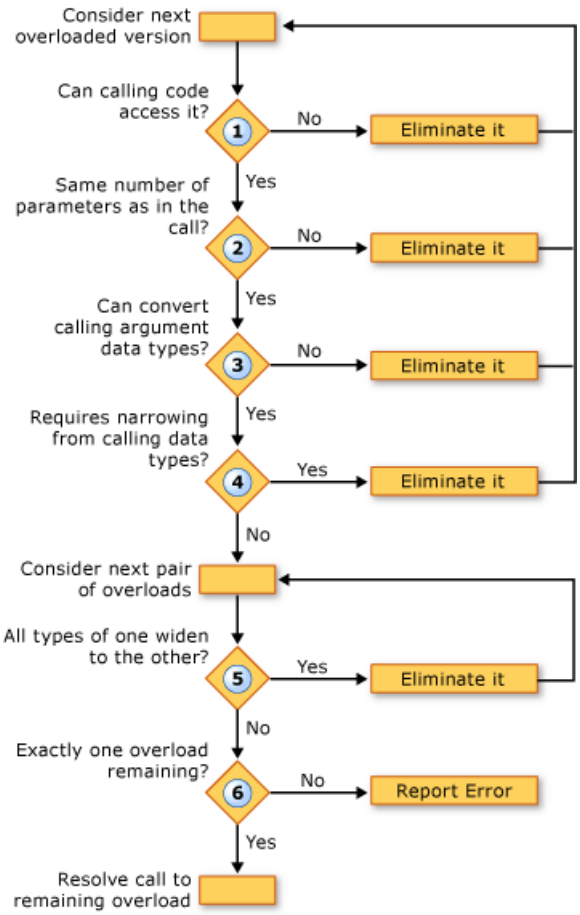
Résolution de surcharge

Lorsque le compilateur Visual Basic rencontre un appel à une procédure défini dans plusieurs versions surchargées, il doit décider de la surcharge à appeler. Pour ce faire, il effectue les étapes suivantes :

1. **Accessibilité.** Il élimine toute surcharge avec un niveau d'accès qui empêche le code appelant de l'appeler.
2. **Nombre de paramètres.** Il élimine les surcharges qui définissent un nombre de paramètres différent de ceux spécifiés dans l'appel.
3. **Types de données des paramètres.** Il élimine les surcharges dont les types de données des arguments d'appel ne sont pas convertibles en types de paramètres définis dans la surcharge.
4. **Conversions restrictives.** Il élimine les surcharges qui nécessitent une conversion restrictive des types des arguments d'appel vers les types des paramètres définis. Cette règle est valable que le commutateur de vérification de type (Option Strict, instruction) ait pour valeur **On** ou **Off**.
5. **Moindre extension.** Il rassemble les surcharges restantes par paires. Pour chaque paire, il compare les types de données des paramètres définis. Si les types de l'une des surcharges s'étendent tous aux types correspondants dans l'autre surcharge, le compilateur élimine la seconde surcharge. En d'autres termes, il conserve la surcharge qui demande l'extension la moins importante.
6. **Candidat unique.** Il continue à rassembler les surcharges par paires jusqu'à ce qu'il ne reste qu'une seule surcharge et il résout l'appel vers celle-ci. Si le compilateur ne peut pas réduire les surcharges vers un seul candidat, il génère une erreur.

L'illustration suivante montre le processus de définition de la version à appeler parmi un jeu de versions surchargées.

Résolution parmi des versions surchargées



L'exemple suivant illustre ce processus de résolution de surcharge :

VB

```
Overloads Sub z(ByVal x As Byte, ByVal y As Double)
```

```
End Sub
Overloads Sub z(ByVal x As Short, ByVal y As Single)
End Sub
Overloads Sub z(ByVal x As Integer, ByVal y As Single)
End Sub
```

VB

```
Dim r, s As Short
Call z(r, s)
Dim p As Byte, q As Short
' The following statement causes an overload resolution error.
Call z(p, q)
```

Dans le premier appel, le compilateur élimine la première surcharge parce que le type du premier argument (**Short**) est restreint au type du paramètre correspondant (**Byte**). Il élimine ensuite la troisième surcharge parce que chaque type d'argument dans la deuxième surcharge (**Short** et **Single**) est étendu au type correspondant dans la troisième surcharge (**Integer** et **Single**). La deuxième surcharge requérant une extension moins importante, le compilateur l'utilise pour l'appel.

Dans le deuxième appel, le compilateur ne peut éliminer aucune des surcharges sur la base de la restriction. Il élimine la troisième surcharge pour la même raison que dans le premier appel, parce qu'il peut appeler la deuxième surcharge moyennant une extension moins importante des types des arguments. Toutefois, le compilateur ne peut pas effectuer de résolution entre la première et la deuxième surcharges. Chaque surcharge possède un type de paramètre défini qui s'étend au type correspondant de l'autre (**Byte** vers **Short**, mais **Single** vers **Double**). Par conséquent, le compilateur génère une erreur de résolution de surcharge.

Arguments Optional et ParamArray de surcharge

Si deux surcharges d'une procédure ont des signatures identiques, mais que le dernier paramètre est déclaré Optional (Visual Basic) dans l'une et ParamArray dans l'autre, le compilateur résout un appel à cette procédure comme suit :

Si l'appel fournit le dernier argument comme	Le compilateur résout l'appel vers la surcharge en déclarant le dernier argument comme
Aucune valeur (argument omis)	**Optional**
Une valeur unique	**Optional**
Au moins deux valeurs dans une liste séparée par des virgules	**ParamArray**
Tableau de toute longueur (y compris un tableau vide)	**ParamArray**

Voir aussi

Tâches

Procédures de dépannage
Comment : définir plusieurs versions d'une procédure
Comment : appeler une procédure surchargée
Comment : surcharger une procédure qui accepte des paramètres optionnels
Comment : surcharger une procédure qui accepte un nombre indéfini de paramètres

Référence

Overloads

Concepts

Paramètres facultatifs
Tableaux de paramètres
Surcharge de procédure
Considérations sur les surcharges de procédures

Procédures de dépannage

Cette page répertorie quelques problèmes courants qui peuvent se produire lors de l'utilisation des procédures.

Retour d'un type tableau à partir d'une procédure Function

Si une procédure **Function** retourne un type de données tableau, vous ne pouvez pas utiliser le nom **Function** pour stocker des valeurs dans les éléments du tableau. Si vous essayez, le compilateur interprète cette tentative comme un appel à **Function**. L'exemple suivant génère des erreurs du compilateur.

```
Function allOnes(ByVal n As Integer) As Integer()
For i As Integer = 1 To n - 1
' The following statement generates a COMPILER ERROR .
allOnes(i) = 1
Next i
' The following statement generates a COMPILER ERROR .
Return allOnes()
End Function
```

L'instruction `allOnes(i) = 1` génère une erreur du compilateur parce qu'elle semble appeler `allOnes` avec un argument de type de données incorrect (un **Integer** singleton au lieu d'un tableau **Integer**). L'instruction `Return allOnes()` génère une erreur du compilateur parce qu'elle semble appeler `allOnes` sans aucun argument.

Approche correcte : pour pouvoir modifier les éléments d'un tableau qui doit être retourné, définissez un tableau interne comme variable locale. L'exemple suivant se compile sans erreur.

VB

```
Function allOnes(ByVal n As Integer) As Integer()
    Dim i As Integer, iArray(n) As Integer
    For i = 0 To n - 1
        iArray(i) = 1
    Next i
    Return iArray
End Function
```

Argument non modifié par un appel de procédure

Si vous envisagez de permettre à une procédure de modifier un élément de programmation sous-jacent d'un argument dans le code appelant, vous devez le passer par référence. Cependant, une procédure peut accéder aux éléments d'un argument de type référence même si vous le passez par valeur.

- **Variable sous-jacente**. Afin de permettre à la procédure de remplacer la valeur de l'élément variable sous-jacent lui-même, la procédure doit déclarer le paramètre ByRef. De plus, le code appelant ne doit pas mettre l'argument entre parenthèses, parce que cela substituerait le mécanisme de passage **ByRef**.
- **Éléments de type référence**. Si vous déclarez un paramètre ByVal, la procédure ne peut pas modifier l'élément variable sous-jacent lui-même. Cependant, si l'argument est un type référence, la procédure peut modifier les membres de l'objet vers lequel elle pointe, même si elle ne peut pas remplacer la valeur de la variable. Par exemple, si l'argument est une variable tableau, la procédure ne peut pas lui assigner un nouveau tableau, mais elle peut modifier un ou plusieurs de ses éléments. Les éléments modifiés sont réfléchis dans la variable tableau sous-jacente du code appelant.

L'exemple suivant permet de définir deux procédures qui acceptent une variable tableau par valeur et opèrent sur ses éléments. La procédure `increase` ajoute simplement la valeur 1 à chaque élément. La procédure `replace` assigne un nouveau tableau au paramètre `a()`, puis ajoute la valeur 1 à chaque élément. Cependant, la réassignation n'affecte pas la variable tableau sous-jacente dans le code appelant, car `a()` est déclaré **ByVal**.

VB

```
Public Sub increase(ByVal a() As Long)
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
```

```
    Next j
End Sub
```

VB

```
Public Sub replace(ByVal a() As Long)
    Dim k() As Long = {100, 200, 300}
    a = k
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

L'exemple suivant permet d'appeler `increase` et `replace`.

VB

```
Dim n() As Long = {10, 20, 30, 40}
Call increase(n)
MsgBox("After increase(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
Call replace(n)
MsgBox("After replace(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
```

Le premier appel **MsgBox** affiche "After increase(n): 11, 21, 31, 41". Parce que `n` est un type référence, `increase` peut modifier ses membres, même s'il est passé **ByVal**.

Le second appel **MsgBox** affiche "After replace(n): 11, 21, 31, 41". Étant donné que `n` est passé **ByVal**, `replace` ne peut pas modifier la variable `n` en lui assignant un nouveau tableau. Lorsque `replace` crée la nouvelle instance de tableau `k` et l'assigne à la variable locale `a`, il perd la référence à `n` passée par le code appelant. Lorsqu'il incrémente les membres de `a`, seul le tableau local `k` est affecté.

Approche correcte : pour pouvoir modifier un élément variable sous-jacent lui-même, passez-le par référence. L'exemple suivant illustre la modification dans la déclaration de `replace` qui lui permet de remplacer un tableau par un autre dans le code appelant.

VB

```
Public Sub replace(ByRef a() As Long)
```

Impossible de définir une surcharge

Si vous souhaitez définir une version surchargée d'une procédure, vous devez utiliser le même nom, mais une signature différente. Si le compilateur ne peut pas différencier votre déclaration d'une surcharge assortie de la même signature, il génère une erreur.

La *signature* d'une procédure est déterminée par la liste de paramètres et le nom de la procédure. Chaque surcharge doit avoir le même nom que toutes les autres surcharges, mais elle doit différer des autres au moins au niveau d'un des autres composants de la signature. Pour plus d'informations, consultez Surcharge de procédure.

Les éléments suivants, bien qu'ils soient relatifs à la liste des paramètres, ne sont pas des composants de la signature d'une procédure :

- Mots clés de modificateur de procédure, tels que **Public**, **Shared** et **Static**
- Noms des paramètres
- Mots clés de modificateur de paramètre, tels que **ByRef** et **Optional**
- Type de données de la valeur de retour (à l'exception d'un opérateur de conversion)

Vous ne pouvez pas surcharger une procédure en faisant varier uniquement un ou plusieurs des éléments précédents.

Approche correcte : pour pouvoir définir une surcharge de procédure, vous devez faire varier la signature. Étant donné vous devez utiliser le même nom, vous devez faire varier le nombre, l'ordre ou les types de données des paramètres. Dans une procédure générique, vous pouvez faire varier le nombre des paramètres de type. Pour un opérateur de conversion (fonction CType), vous pouvez faire varier le type de retour.

Résolution de surcharge avec des arguments Optional et ParamArray

Si vous surchargez une procédure avec un ou plusieurs paramètres Optional (Visual Basic) ou un paramètre ParamArray, vous devez éviter de dupliquer une des *surcharges implicites*. Pour plus d'informations, consultez Considérations sur les surcharges de procédures.

Appel d'une version incorrecte d'une procédure surchargée

Si une procédure dispose de plusieurs versions surchargées, vous devez connaître toutes leurs listes de paramètres et comprendre comment Visual Basic résout des appels parmi les surcharges, faute de quoi vous risquez d'appeler une surcharge qui n'est pas celle voulue.

Lorsque vous avez identifié la surcharge à appeler, veillez à observer les règles suivantes :

- Fournissez le nombre correct d'arguments dans le bon ordre.
- Dans l'idéal, vos arguments doivent disposer exactement des mêmes types de données que les paramètres correspondants. Dans tous les cas, le type de données de chaque argument doit être étendu à celui de son paramètre correspondant. Cette règle est valable même lorsque la valeur **Off** est affectée à l'instruction Option Strict. Si une surcharge requiert une conversion restrictive de votre liste d'arguments, cette surcharge ne peut pas être appelée.
- Si vous fournissez des arguments qui requièrent l'extension, rendez leurs types de données aussi proches que possible des types de données des paramètres correspondants. Si plusieurs surcharges acceptent les types de données de vos arguments, le compilateur résout votre appel en fonction de la surcharge qui exige l'extension la moins importante.

Vous pouvez réduire le risque d'incompatibilité de type de données en utilisant le mot clé de conversion de la fonction CType lors de la préparation de vos arguments.

Échec de la résolution de surcharge

Lorsque vous appelez une procédure surchargée, le compilateur tente de conserver une seule surcharge. S'il y parvient, il résout l'appel à cette surcharge. S'il élimine toutes les surcharges ou ne parvient pas à réduire les surcharges admissibles à un seul candidat, il génère une erreur.

L'exemple suivant illustre le processus de résolution de surcharge.

VB

```
Overloads Sub z(ByVal x As Byte, ByVal y As Double)
End Sub
Overloads Sub z(ByVal x As Short, ByVal y As Single)
End Sub
Overloads Sub z(ByVal x As Integer, ByVal y As Single)
End Sub
```

VB

```
Dim r, s As Short
Call z(r, s)
Dim p As Byte, q As Short
' The following statement causes an overload resolution error.
Call z(p, q)
```

Dans le premier appel, le compilateur élimine la première surcharge parce que le type du premier argument (**Short**) est restreint au type du paramètre correspondant (**Byte**). Il élimine ensuite la troisième surcharge parce que chaque type d'argument dans la deuxième surcharge (**Short** et **Single**) est étendu au type correspondant dans la troisième surcharge (**Integer** et **Single**). La deuxième surcharge requérant une extension moins importante, le compilateur l'utilise pour l'appel.

Dans le deuxième appel, le compilateur ne peut éliminer aucune des surcharges sur la base de la restriction. Il élimine la troisième surcharge pour la même raison que dans le premier appel, parce qu'il peut appeler la deuxième surcharge moyennant une extension moins importante des types des arguments. Toutefois, le compilateur ne peut pas effectuer de

résolution entre la première et la deuxième surcharges. Chaque surcharge possède un type de paramètre défini qui s'étend au type correspondant de l'autre (**Byte** vers **Short**, mais **Single** vers **Double**). Par conséquent, le compilateur génère une erreur de résolution de surcharge.

Approche correcte : pour pouvoir appeler une procédure surchargée sans ambiguïté, utilisez la fonction CType pour faire correspondre les types de données des arguments aux types des paramètres. L'exemple suivant illustre un appel à `z` qui force la résolution sur la deuxième surcharge.

VB

```
Call z(CType(p, Short), CType(q, Single))
```

Résolution de surcharge avec des arguments Optional et ParamArray

Si deux surcharges d'une procédure ont des signatures identiques, mais que le dernier paramètre est déclaré Optional (Visual Basic) dans l'une et ParamArray dans l'autre, le compilateur résout un appel à cette procédure d'après la correspondance la plus proche. Pour plus d'informations, consultez Résolution de surcharge.

Voir aussi

Concepts

Procédures dans Visual Basic
Sub, procédures
Function, procédures
Procédures Property
Procédures d'opérateur
Paramètres et arguments d'une procédure
Surcharge de procédure
Considérations sur les surcharges de procédures
Résolution de surcharge

Flux de contrôle en Visual Basic

Lorsqu'un programme n'est pas réglé, il traite ses instructions du début à la fin. L'écriture de certains programmes très simples peut comprendre uniquement ce flux unidirectionnel. Toutefois, la puissance et l'utilité d'un langage de programmation proviennent principalement de la possibilité de modifier l'ordre d'exécution grâce à des instructions de contrôle et à des boucles.

Les structures de contrôle permettent de réguler le flux de l'exécution de votre programme. Elles vous permettent d'écrire du code Visual Basic qui prend des décisions ou qui répète des actions. D'autres structures de contrôle vous permettent de garantir la mise à disposition d'une ressource ou d'exécuter une série d'instructions sur la même référence d'objet.

Dans cette section

Structures de décision

Décrit les structures de contrôle utilisées pour la création de branche.

Structures de boucle

Présente les structures de contrôle utilisées pour répéter des processus.

Autres structures de contrôle

Décrit les structures de contrôle utilisées pour la mise à disposition de ressources et l'accès aux objets.

Structures de contrôle imbriquées

Présente les structures de contrôle contenues dans d'autres structures de contrôle.

Comment : transférer un contrôle hors d'une structure de contrôle

Récapitule le transfert direct depuis une structure de contrôle.

Rubriques connexes

Instruction de contrôle pour les utilisateurs de Visual Basic 6.0

Répertorie les modifications de comportement et de syntaxe des structures de contrôle à partir des versions précédentes de Visual Basic.

Liste des flux de contrôle

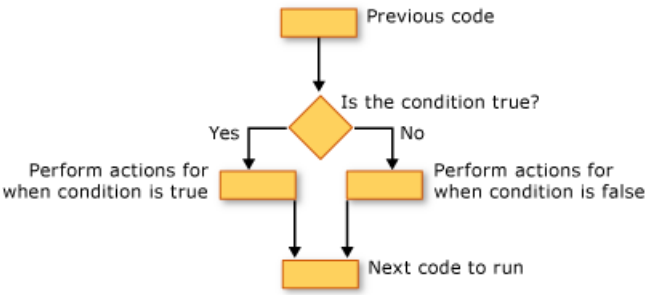
Propose des liens vers des pages de référence du langage à ce sujet.

Structures de décision

Visual Basic vous permet de tester des conditions et d'effectuer diverses opérations selon les résultats de ce test. Vous pouvez tester une condition vraie ou fausse, plusieurs valeurs d'une expression ou plusieurs exceptions générées lors de l'exécution d'un ensemble d'instructions.

L'illustration suivante représente une structure de décision qui recherche une condition ayant la valeur true et exécute différentes actions selon qu'elle a la valeur true ou false.

Exécution d'actions différentes selon qu'une condition a la valeur true ou false



If...Then...Else, construction

Les constructions **If...Then...Else** vous permettent de tester une ou plusieurs conditions et d'exécuter une ou plusieurs instructions selon chaque condition. Vous pouvez tester des conditions et prendre des mesures de la manière suivante :

- Exécutez une ou plusieurs instructions si une condition est **True**
- Exécutez une ou plusieurs instructions si une condition est **False**
- Exécutez des instructions si une condition est **True** et d'autres si elle est **False**
- Testez une condition supplémentaire si une condition antérieure est **False**

La structure de contrôle qui offre toutes ces possibilités est If...Then...Else, instruction (Visual Basic). Vous pouvez utiliser une version sur une ligne si vous n'avez qu'un test et qu'une instruction à exécuter. Si vous avez un jeu plus complexe de conditions et d'actions, vous pouvez utiliser la version multiligne.

Select...Case, construction

La construction **Select...Case** vous permet d'évaluer une expression une fois et d'exécuter différents jeux d'instructions selon différentes valeurs possibles. Pour plus d'informations, consultez Select...Case, instruction (Visual Basic).

Try...Catch...Finally, construction

Les constructions **Try...Catch...Finally** vous permettent d'exécuter un jeu d'instructions sous un environnement qui conserve le contrôle si l'une de vos instructions entraîne une exception. Vous pouvez prendre des mesures différentes pour des exceptions différentes. Vous pouvez éventuellement spécifier un bloc de code à exécuter avant de sortir de la construction **Try...Catch...Finally** entière, quelle qu'en soit l'issue. Pour plus d'informations, consultez Try...Catch...Finally, instruction (Visual Basic).

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter des instructions en fonction d'une ou de plusieurs conditions
Comment : tester plusieurs valeurs d'une expression
Comment : conserver le contrôle si une erreur se produit

Concepts

Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : exécuter des instructions en fonction d'une ou de plusieurs conditions

L'If...Then...Else, instruction (Visual Basic) permet d'exécuter une instruction ou un ensemble d'instructions spécifique en fonction de la valeur Boolean, type de données (Visual Basic) d'une condition. La condition provient généralement d'une comparaison entre deux valeurs, mais elle peut être une expression qui évalue une valeur **Boolean** (**True** ou **False**). Elle peut comprendre des valeurs d'autres types de données, tels que des types numériques, qui ont été converties en type **Boolean**.

Pour exécuter une ou plusieurs instructions si une condition a la valeur True

- Si vous n'avez qu'une seule instruction à exécuter, utilisez la syntaxe sur une ligne de la construction **If**...**Then**...**Else**. Les instructions **Else** ou **End If** ne sont pas nécessaires. L'exemple suivant illustre ce comportement :

```
Sub fixDate()
    Dim myDate As Date = #2/13/1973#
    If myDate < Now Then myDate = Now
End Sub
```

- ou -

- Pour exécuter plusieurs lignes de code lorsque la condition est **True**, vous devez utiliser une syntaxe sur plusieurs lignes, comprenant l'instruction **End If**. Si vous n'avez aucun code à exécuter lorsque la condition est **False**, vous omettez l'instruction **Else**. L'exemple suivant illustre ce comportement :

```
Dim alertLabel As New System.Windows.Forms.Label
Sub alertUser(ByVal value As Long)
    If value = 0 Then
        alertLabel.ForeColor = System.Drawing.Color.Red
        alertLabel.Font = New Font(alertLabel.Font, _
            FontStyle.Bold Or FontStyle.Italic)
    End If
End Sub
```

Pour exécuter des instructions si une condition a la valeur True et d'autres si elle a la valeur False

- Utilisez une construction **If**...**Then**...**Else** avec l'instruction Else (Visual Basic) pour définir deux blocs d'instructions. Visual Basic exécute un bloc si la condition est **True** et l'autre si elle est **False**. L'exemple suivant illustre ce comportement :

```
Dim alertLabel As New System.Windows.Forms.Label
Sub alertUser(ByVal value As Long)
    If value = 0 Then
        alertLabel.ForeColor = System.Drawing.Color.Red
        alertLabel.Font = New Font(alertLabel.Font, _
            FontStyle.Bold Or FontStyle.Italic)
    Else
        alertLabel.Forecolor = System.Drawing.Color.Black
        alertLabel.Font = New Font(alertLabel.Font, _
            FontStyle.Regular)
    End If
End Sub
```

Pour tester des conditions supplémentaires si la première condition a la valeur False

- Utilisez une construction **If**...**Then**...**Else** avec une ou plusieurs instructions ElseIf (Visual Basic) pour tester des conditions supplémentaires si la première condition est **False**. Dans l'exemple suivant, la procédure **Function** calcule une prime

salariale en fonction du taux de performance. Le bloc d'instruction qui suit l'instruction **Else** s'exécute uniquement si les conditions des instructions **If** et **ElseIf** sont toutes **False**.

```
Function bonus(ByVal performance As Integer, ByVal salary As Decimal) _
    As Decimal
    If performance = 1 Then
        Return salary * 0.1
    ElseIf performance = 2 Then
        Return salary * 0.09
    ElseIf performance = 3 Then
        Return salary * 0.07
    Else
        Return 0
    End If
End Function
```

Visual Basic teste les conditions dans l'ordre dans lequel elles apparaissent dans les instructions **If**...**Then**...**Else**. Lorsqu'il rencontre une condition **True** ou une instruction **Else**, il exécute le bloc d'instruction correspondant. Le contrôle passe alors à l'instruction qui suit l'instruction **End If**.

Vous pouvez avoir autant d'instructions **ElseIf** que nécessaire, voire aucune. Vous pouvez inclure ou omettre une instruction **Else**, que vous ayez ou non des instructions **ElseIf**.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : tester plusieurs valeurs d'une expression
Comment : conserver le contrôle si une erreur se produit

Concepts

Structures de décision
Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : tester plusieurs valeurs d'une expression

Si vous comparez la même expression à plusieurs valeurs différentes, vous pouvez utiliser la construction Select...Case, instruction (Visual Basic) plutôt que la construction If...Then...Else, instruction (Visual Basic). Tandis que les instructions **If** et ElseIf (Visual Basic) permettent d'évaluer une expression différente dans chaque instruction, l'instruction **Select** évalue une expression unique une seule fois et l'utilise pour chaque comparaison.

Pour évaluer une expression une fois et tester plusieurs valeurs

- Utilisez la construction **Select**...**Case** pour spécifier l'expression et les valeurs à tester. Chaque instruction **Case** peut contenir une ou plusieurs valeurs, une plage de valeurs ou une combinaison de valeurs et d'opérateurs de comparaison. Vous pouvez utiliser une instruction **Case Else** pour gérer toutes les valeurs non testées par les instructions **Case** précédentes. L'exemple de code suivant illustre ces possibilités.

```
Function bonus(ByVal performance As Integer, ByVal salary As Decimal) _
    As Decimal
    Select performance
        Case 1
            Return salary * 0.1
        Case 2, 3
            Return salary * 0.09
        Case 5 To 7
            Return salary * 0.07
        Case 4, 8 To 10
            Return salary * 0.05
        Case Is < 15
            Return 100
        Case Else
            Return 0
   End Select
End Function
```

  Visual Basic compare la valeur de l'expression aux valeurs des instructions **Case** selon leur ordre d'apparition dans la construction **Select**...**Case**. S'il trouve une correspondance ou une instruction **Case Else**, il exécute le bloc d'instructions correspondant. Dans tous les cas, l'exécution passe ensuite à l'instruction suivant l'instruction **End Select**.

  Vous pouvez avoir un nombre quelconque d'instructions **Case** et vous pouvez inclure ou omettre une instruction **Case Else** même s'il n'y a aucune instruction **Case**.

Un exemple de code pour la construction **Select**...**Case** est également disponible en tant qu'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter des instructions en fonction d'une ou de plusieurs conditions
Comment : conserver le contrôle si une erreur se produit

Concepts

Structures de décision
Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : conserver le contrôle si une erreur se produit

Vous pouvez utiliser la construction Try...Catch...Finally, instruction (Visual Basic) pour la *gestion structurée des exceptions*. Vous pouvez alors exécuter un bloc d'instructions particulier si une exception spécifiée se produit lors de l'exécution du code. Dans ce cas, le code *lève* l'exception que vous *interceptez* avec l'instruction **Catch** adéquate.

Pour exécuter un jeu d'instructions si votre code provoque une exception

- Utilisez la construction **Try**...**Catch**... **Finally** pour y inclure le code qui peut provoquer une exception. Spécifiez ensuite le code à exécuter si une exception se produit, et fournissez éventuellement un jeu d'instructions à exécuter avant que le contrôle ne quitte la construction **Try**...**Catch**... **Finally**.

  L'exemple suivant tente de calculer la date et l'heure exactes 100 ans après la valeur fournie dans la variable **Object** de `givenDate`.

```
Dim givenDate As Object
Dim nextCentury As Date
Try
    nextCentury = Microsoft.VisualBasic.DateAdd("yyyy", 100, givenDate)
    Catch thisExcep As System.ArgumentOutOfRangeException
    ' The resulting date/time is later than December 31, 9999.
    Catch thisExcep As System.ArgumentException
    ' At least one argument has an invalid value.
    Catch thisExcep As System.InvalidCastException
    ' The value in givenDate cannot be interpreted as a date/time.
    Catch
    ' An unforeseen exception has occurred.
    Finally
    ' This block is always run before leaving the Try structure.
End Try
```

  Les trois premiers blocs **Catch** gèrent les exceptions que vous pouvez anticiper à partir de DateAdd, fonction (Visual Basic). Vous pouvez traiter n'importe quelle exception inattendue dans le dernier bloc **Catch**.

  Dans tous les cas, le bloc **Finally** est toujours le dernier code exécuté avant de quitter la construction **Try**...**Catch**...**Finally**. Si vous créez ou ouvrez des ressources, telles que des objets ou des connexions de base de données dans un bloc **Try** ou **Catch**, vous pouvez utiliser le bloc **Finally** pour les fermer et les supprimer, le cas échéant.

  Si la variable d'exception `thisExcep` n'apparaît pas dans une instruction de déclaration, telle que **Dim**, l'instruction **Catch** avec la clause **As** est utilisée comme une déclaration.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter des instructions en fonction d'une ou de plusieurs conditions
Comment : tester plusieurs valeurs d'une expression

Concepts

Structures de décision
Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

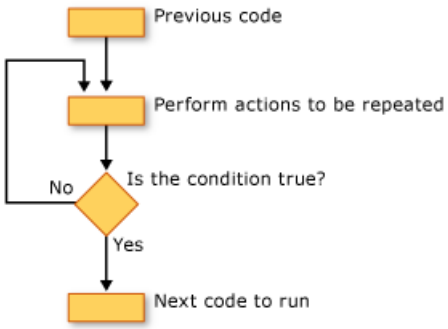
Flux de contrôle en Visual Basic

Structures de boucle

Les structures de boucle Visual Basic vous permettent d'exécuter continuellement une ou plusieurs lignes de code. Vous pouvez répéter les instructions d'une structure de boucle jusqu'à ce qu'une condition soit **True**, jusqu'à ce qu'une condition soit **False**, un certain nombre de fois ou une seule fois pour chaque élément d'une collection.

L'illustration suivante représente une structure de boucle qui exécute un jeu d'instructions jusqu'à ce qu'une condition prenne la valeur true.

Exécution d'un jeu d'instructions jusqu'à ce qu'une condition prenne la valeur true



Boucles While

La construction **While**...**End While** exécute un jeu d'instructions tant que la condition spécifiée dans l'instruction **While** est **True**. Pour plus d'informations, consultez While...End While, instruction (Visual Basic).

Boucles Do

La construction **Do**...**Loop** vous permet de tester une condition au début ou à la fin d'une structure de boucle. Vous pouvez également faire en sorte de répéter la boucle tant que la condition reste **True** ou jusqu'à ce qu'elle devienne **True**. Pour plus d'informations, consultez Do...Loop, instruction (Visual Basic).

Boucles For

La construction **For**...**Next** exécute la boucle un certain nombre de fois. Elle utilise une variable de contrôle de boucle, également nommée *compteur*, pour assurer le suivi des répétitions. Vous spécifiez les valeurs de début et de fin de ce compteur et éventuellement la valeur d'incrémentation d'une répétition à la suivante. Pour plus d'informations, consultez For...Next, instruction (Visual Basic).

Boucles For Each

La construction **For Each**...**Next** exécute un jeu d'instructions une fois pour chaque élément d'une collection. Vous spécifiez la variable de contrôle de boucle, mais il n'est pas nécessaire de déterminer la valeur de début ou de fin. Pour plus d'informations, consultez For Each...Next, instruction (Visual Basic).

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter plusieurs instructions à plusieurs reprises
Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau
Comment : améliorer les performances d'une boucle
Comment : passer à l'itération suivante d'une boucle

Concepts

Structures de décision
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : exécuter plusieurs instructions à plusieurs reprises

Vous pouvez utiliser des structures de boucle pour exécuter à plusieurs reprises un bloc d'instructions. La boucle peut s'exécuter un nombre indéterminé de fois en fonction de la valeur **Boolean** d'une condition ou un nombre défini de fois en fonction d'une variable spéciale.

Bouclage d'un nombre indéterminé de fois

Pour exécuter un groupe d'instructions tant qu'une condition a la valeur True

- Utilisez While...End While, instruction (Visual Basic) pour spécifier la condition qui contrôle la répétition de la boucle. L'exemple suivant répète le bloc d'instructions tant que `number` est supérieur à 6.

```
Sub checkWhile()
    Dim counter As Integer = 0
    Dim number As Integer = 10
    While number > 6
        number -= 1
        counter += 1
    End While
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

  L'instruction **While** vérifie toujours la condition avant de commencer la boucle. Si `number` a été initialisé à 6 au lieu de 10, les instructions à l'intérieur de la boucle ne sont jamais exécutées.

Pour exécuter un groupe d'instructions pendant qu'une condition conserve la valeur True

- Utilisez Do...Loop, instruction (Visual Basic) et spécifiez la condition de test soit au début, soit à la fin de la boucle. Le positionnement du mot clé While (Visual Basic) détermine l'emplacement où la condition est testée. L'exemple suivant illustre ce comportement :

```
Sub checkWhileFirst()
    Dim counter As Integer = 0
    Dim number As Integer = 10
    Do While number > 6
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
Sub checkWhileLast()
    Dim counter As Integer = 0
    Dim number As Integer = 5
    Do
        number -= 1
        counter += 1
    Loop While number > 6
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

  Dans l'exemple précédent, la première boucle **Do** s'exécute quatre fois et la deuxième boucle **Do** s'exécute une fois.

Pour exécuter un groupe d'instructions jusqu'à ce qu'une condition prenne la valeur True

- Utilisez la construction **Do**...**Loop** avec le mot clé Until au lieu de **While**. Comme pour **While**, le positionnement du mot clé détermine l'emplacement où la condition est testée. L'exemple suivant illustre ce comportement :

```
Sub checkUntilFirst()
    Dim counter As Integer = 0
    Dim number As Integer = 20
    Do Until number = 15
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
Sub checkUntilLast()
    Dim counter As Integer = 0
    Dim number As Integer = 20
    Do
        number -= 1
        counter += 1
    Loop Until number = 15
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

Dans l'exemple précédent, chaque boucle **Do** s'exécute cinq fois.

Bouclage d'un nombre défini de fois

Les boucles **While** et **Do** sont d'une utilisation pratique lorsque vous ne connaissez pas à l'avance le nombre de fois où les instructions doivent être exécutées dans une boucle. Toutefois, si vous devez exécuter la boucle un certain nombre de fois, il est préférable d'utiliser For...Next, instruction (Visual Basic). Contrairement à une boucle **While** ou **Do**, une boucle **For**...**Next** utilise une *variable de contrôle* dont la valeur augmente ou diminue lors de chaque répétition de la boucle.

Pour exécuter un groupe d'instructions un nombre défini de fois

1. Déterminez les valeurs de début et de fin de la variable de contrôle, et utilisez l'instruction **For** pour les spécifier.

   ```
   For i As Integer = 1 To 10
   ```

   Si la variable de contrôle n'est pas déclarée hors de la boucle, vous pouvez utiliser la clause **As** pour la déclarer dans le cadre de l'instruction **For**.

2. Utilisez le mot clé Step pour indiquer de combien la variable de contrôle doit augmenter pour chaque itération. Elle augmente de 1, sauf indication contraire. Utilisez une valeur négative pour faire diminuer la variable de contrôle.

   ```
   For i As Integer = 10 To 1 Step -1
   ```

3. Complétez la construction **For**...**Next** avec une instruction **Next** qui suit la dernière instruction à répéter. Vous pouvez spécifier la variable de contrôle dans l'instruction **Next**.

   ```
   Function addBackward(ByVal highest As Integer) As Integer
       Dim total As Integer = 0
       For i As Integer = highest To 1 Step -1
           total += i
       Next i
       Return total
   End Function
   ```

   L'exemple précédent retourne la somme de tous les nombres entiers de 1 jusqu'à la valeur passée au paramètre `highest`.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle

Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau
Comment : améliorer les performances d'une boucle
Comment : passer à l'itération suivante d'une boucle

Référence

For Each...Next, instruction (Visual Basic)

Concepts

Structures de décision
Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau

La construction d'instruction **For Each** est semblable à la boucle **For**...**Next**, mais elle exécute le bloc d'instructions de chaque élément d'une collection, et non un nombre spécifié de fois. Une boucle **For Each**...**Next** utilise une *variable d'élément* qui représente un élément différent de la collection durant chaque répétition de la boucle.

Répétition d'un bloc d'instructions pour une collection

Pour exécuter un groupe d'instructions pour chaque élément d'une collection

1. Identifiez la collection sur laquelle exécuter le bloc d'instructions et utilisez l'For Each...Next, instruction (Visual Basic) pour spécifier à la fois la variable d'élément et la collection.

   ```
                   For Each thisControl As System.Windows.Forms.Control In thisForm.Contr
   ols
   ```

   Si la variable d'élément n'est pas déclarée hors de la boucle, vous pouvez utiliser la clause **As** pour la déclarer dans le cadre de l'instruction **For Each**.

   Visual Basic substitue automatiquement un élément différent de la collection à chaque itération.

2. Complétez la construction **For Each**...**Next** avec une instruction **Next** après la dernière instruction à répéter. Vous pouvez spécifier la variable d'élément dans l'instruction **Next**.

   ```
   Sub lightBlueBackground(ByVal thisForm As System.Windows.Forms.Form)
       For Each thisControl As System.Windows.Forms.Control In thisForm.Controls
           thisControl.BackColor = System.Drawing.Color.LightBlue
       Next thisControl
   End Sub
   ```

   Vous pouvez lire les éléments de la collection, mais vous ne pouvez pas apporter de modifications à la collection elle-même, telle qu'ajouter, supprimer ou remplacer un élément. Toutefois, si un élément est un type référence, vous pouvez accéder et définir ses membres. Chaque élément Control étant un type référence, le code de l'exemple précédent peut modifier sa propriété BackColor.

Répétition d'un bloc d'instructions pour un tableau

Vous pouvez également parcourir un tableau avec une boucle **For Each**...**Next**. Toutefois, comme avec les collections, vous pouvez uniquement lire les éléments du tableau, et non les modifier.

Pour exécuter un groupe d'instructions pour chaque élément d'un tableau

1. Utilisez l'instruction **For Each** pour spécifier à la fois la variable d'élément et le tableau. Ne faites pas suivre le nom du tableau de parenthèses.

   Visual Basic traite le tableau de la même manière qu'une collection.

2. Faites suivre le bloc d'instructions d'une instruction **Next**. Vous pouvez spécifier la variable d'élément dans l'instruction **Next**.

   La procédure ci-après a pour but de trouver la somme des éléments d'un tableau et de réinitialiser chaque élément à zéro.

   ```
   Function sumAndReset(ByRef numbers() As Integer) As Integer
       Dim sum As Integer = 0
       For Each elt As Integer In numbers
           sum += elt
   ```

```
        ' The following statement works only on the local copy
        ' of the array, not on the original array.
        elt = 0
    Next elt
    Return sum
End Function
```

Le calcul de la somme fonctionne comme prévu, car il ne repose que sur la lecture des éléments. En revanche, la réinitialisation à zéro ne fonctionne pas comme prévu, car seule la copie locale de chaque élément est réinitialisée à zéro, tandis que l'élément d'origine reste inchangé dans le tableau d'origine.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter plusieurs instructions à plusieurs reprises
Comment : améliorer les performances d'une boucle
Comment : passer à l'itération suivante d'une boucle

Référence

For...Next, instruction (Visual Basic)

Concepts

Structures de décision
Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : améliorer les performances d'une boucle

Vous pouvez optimiser les performances d'une boucle en utilisant les types de données les plus efficaces. Pour une boucle courte qui ne s'exécute pas plusieurs fois, la différence peut être négligeable. Toutefois, si la boucle s'exécute un grand nombre de fois, vous pouvez protéger les performances de manière considérable.

Integer et **UInteger** sont les types les plus efficaces sur les plates-formes actuelles. **Short**, **Long**, **UShort** et **ULong** ne sont pas aussi efficaces, et **Decimal** est considérablement plus lent. Pour plus d'informations, consultez Types de données numériques.

Pour optimiser les performances d'une boucle For...Next

- Utilisez le type de données le plus efficace pour la variable de contrôle. L'exemple suivant illustre certaines variations possibles sur une boucle. Les minutages absolus dépendent de la plate-forme, mais les comparaisons à l'exécution restent valides.

```
For fastest As Integer = 0 to 1000000
    ' Insert statements to execute for each value of fastest.
Next fastest
For notAsFast As Long = 0 to 1000000
    ' Insert statements to execute for each value of notAsFast.
Next notAsFast
For muchSlower As Decimal = 0 to 1000000
    ' Insert statements to execute for each value of muchSlower.
Next muchSlower
```

  Le premier cas s'exécute un peu plus rapidement que le second. Toutefois, **Integer** peut gérer des valeurs jusqu'à 2 147 483 647 et **UInteger** jusqu'à 4 294 967 295 uniquement. Les deuxième et troisième cas peuvent gérer des valeurs plus grandes puisque **Long** et **Decimal** acceptent une plus large gamme d'entiers, mais ils s'exécutent plus lentement. Il vous faudra peut-être prendre une décision de conception et choisir entre la vitesse et la capacité des types de données que vous utilisez.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter plusieurs instructions à plusieurs reprises
Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau
Comment : passer à l'itération suivante d'une boucle

Référence

For...Next, instruction (Visual Basic)

Concepts

Structures de boucle

Autres ressources

Flux de contrôle en Visual Basic

Comment : passer à l'itération suivante d'une boucle

Si vous avez terminé le traitement de l'itération actuelle d'une boucle **Do**, **For** ou **While**, vous pouvez passer immédiatement à l'itération suivante à l'aide de Continue, instruction (Visual Basic).

Passage à l'itération suivante

Pour passer à l'itération suivante d'une boucle For...Next

1. Écrivez la boucle **For**...**Next** de manière normale.
2. Utilisez **Continue For** sur tous les emplacements où vous souhaitez terminer l'itération actuelle et passer immédiatement à l'itération suivante.

```
Public Function findLargestRatio(ByVal high() As Double, _
    ByVal low() As Double) As Double
    Dim ratio As Double
    Dim largestRatio As Double = Double.MinValue
    For counter As Integer = 0 To low.GetUpperBound(0)
        If Math.Abs(low(counter)) < System.Double.Epsilon _
            Then Continue For
        ratio = high(counter) / low(counter)
        If Double.IsInfinity(ratio) OrElse Double.IsNaN(ratio) _
            Then Continue For
        If ratio > largestRatio Then largestRatio = ratio
    Next counter
    Return largestRatio
End Function
```

Passage à partir d'une boucle imbriquée

Si vous avez des boucles **Do**, **For** ou **While** imbriquées les unes dans les autres, vous pouvez passer immédiatement à l'itération suivante, quel que soit le niveau dans l'imbrication. Toutefois, cela est vrai uniquement lorsque les boucles sont de types différents. Si vous avez des boucles imbriquées du même type, par exemple des boucles **While** imbriquées, **Continue While** passe à l'itération suivante de la boucle **While** la plus profonde.

Pour passer à l'itération suivante d'une boucle Do à partir d'une boucle For imbriquée

1. Écrivez les boucles imbriquées de manière normale.
2. Utilisez **Continue Do** sur tous les emplacements où vous souhaitez terminer l'itération actuelle de la boucle **For** interne et passer à l'itération suivante de la boucle **Do** externe.

```
Public Sub divideElements(ByRef matrix(,) As Double)
    Dim i As Integer = -1
    Do Until i > matrix.GetUpperBound(0)
        i += 1
        For j As Integer = 0 To matrix.GetUpperBound(1)
            If matrix(j, j) = 0 Then Continue Do
            matrix(i, j) /= matrix(j, j)
        Next j
    Loop
End Sub
```

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter plusieurs instructions à plusieurs reprises

Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau
Comment : améliorer les performances d'une boucle

Référence

While...End While, instruction (Visual Basic)
Do...Loop, instruction (Visual Basic)
For...Next, instruction (Visual Basic)

Concepts

Structures de boucle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Autres structures de contrôle

Visual Basic fournit des structures de contrôle qui vous permettent de supprimer une ressource ou de réduire le nombre de fois où vous devez répéter une référence d'objet.

Utilisation de la construction Using...End

La construction **Using...End Using** établit un bloc d'instructions dans lequel vous utilisez une ressource telle qu'une connexion SQL. Vous pouvez éventuellement acquérir la ressource avec l'instruction **Using**. Lorsque vous quittez le bloc **Using**, Visual Basic supprime automatiquement la ressource afin qu'un autre code puisse l'utiliser. La ressource doit être locale et supprimable. Pour plus d'informations, consultez Using, instruction (Visual Basic).

With...End With, construction

La construction **With...End With** vous permet de spécifier une référence d'objet une fois, puis d'exécuter une série d'instructions qui accèdent à ses membres. Vous pouvez ainsi simplifier votre code et améliorer les performances puisqu'il n'est pas nécessaire que Visual Basic rétablisse la référence pour chaque instruction qui accède au code. Pour plus d'informations, consultez With...End With, instruction (Visual Basic).

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle

Référence

Using, instruction (Visual Basic)

With...End With, instruction (Visual Basic)

Concepts

Structures de décision

Structures de boucle

Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : supprimer une ressource système

Vous pouvez utiliser un bloc **Using** pour vous assurer que le système supprime une ressource lorsque votre code quitte le bloc. Cette opération est utile si vous utilisez une ressource système qui consomme une grande quantité de mémoire ou que d'autres composants souhaitent également utiliser.

Pour supprimer une connexion de base de données lorsque votre code a fini de l'utiliser

1. Assurez-vous d'inclure l'instruction Imports appropriée pour la connexion de base de données au début de votre fichier source (dans le cas présent, System.Data.SqlClient).

2. Créez un bloc **Using** avec les instructions **Using** et **End Using** À l'intérieur du bloc, insérez le code qui traite de la connexion de base de données.

3. Déclarez la connexion et créez-en une instance dans le cadre de l'instruction **Using**.

```
' Insert the following line at the beginning of your source file.
Imports System.Data.SqlClient
Public Sub AccessSql(ByVal s As String)
    Using sqc As New System.Data.SqlClient.SqlConnection(s)
        MsgBox("Connected with string """ & sqc.ConnectionString & """")
    End Using
End Sub
```

   Le système supprime la ressource quelle que soit la manière dont vous quittez le bloc, y compris en cas d'exception non gérée.

   Notez que vous ne pouvez pas accéder à `sqc` à l'extérieur du bloc **Using**, car sa portée est limitée au bloc.

   Vous pouvez utiliser cette même technique sur une ressource système, telle qu'un handle de fichier ou un wrapper COM. Vous utilisez un bloc **Using** lorsque vous souhaitez être sûr de laisser la ressource à la disposition d'autres composants après avoir quitté le bloc **Using**.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle
Comment : exécuter plusieurs actions sur un objet

Référence

Using, instruction (Visual Basic)
SqlConnection

Concepts

Structures de décision
Structures de boucle
Autres structures de contrôle
Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Comment : exécuter plusieurs actions sur un objet

Dans Visual Basic, vous devez généralement spécifier un objet dans chaque instruction qui appelle l'une de ses méthodes ou qui accède à l'une de ses propriétés. Toutefois, si un ensemble d'instructions fonctionne sur le même objet, vous pouvez utiliser une structure **With**...**End With** pour spécifier une seule fois l'objet pour toutes les instructions. De cette façon, vos procédures s'exécutent plus vite et vous ne devez pas retaper de texte.

Exemple

L'exemple suivant définit la couleur de premier plan et le style de police d'un Label en fonction de la valeur d'un argument de procédure.

```
Imports draw = System.Drawing
' The preceding statement must appear at the beginning of the source file.
Dim alertLabel As New System.Windows.Forms.Label
Sub alertUser(ByVal value As Long)
    With alertLabel
        If value = 0 Then
            .ForeColor = draw.Color.Red
            .Font = New draw.Font(.Font, draw.FontStyle.Bold Or draw.FontStyle.Italic)
        Else
            .Forecolor = draw.Color.Black
            .Font = New draw.Font(.Font, draw.FontStyle.Regular)
        End If
    End With
End Sub
```

Notez l'utilisation de Or, opérateur (Visual Basic) pour associer des styles de police. Cette utilisation spécifie la combinaison souhaitée d'indicateurs binaires. And, opérateur (Visual Basic) aurait produit 0 parce que tous les membres d'énumération FontStyle utilisent des bits différents.

Notez également l'utilisation de Imports, instruction pour définir l'alias d'importation `draw` qui raccourcit et simplifie la lecture de chaque référence aux membres `System.Drawing`.

Vous pouvez également imbriquer des instructions **With...End With** les unes à l'intérieur des autres, comme dans le code suivant :

VB

```
Sub setupForm()
    Dim anotherForm As New System.Windows.Forms.Form
    Dim button1 As New System.Windows.Forms.Button
    With anotherForm
        .Show()
        .Top = 250
        .Left = 250
        .ForeColor = System.Drawing.Color.LightBlue
        .BackColor = System.Drawing.Color.DarkBlue
        .Controls.Add(button1)
        With .Controls.Item(1)
            .BackColor = System.Drawing.Color.Thistle
            .Text = "Text on button1"
        End With
    End With
End Sub
```

Toutefois, à l'intérieur de l'instruction **With**, la syntaxe fait référence à l'objet imbriqué ; les propriétés de l'objet de l'instruction **With** extérieure ne sont pas définies.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle

Comment : supprimer une ressource système

Comment : accélérer l'accès à un objet comportant un chemin d'accès de qualification long

Référence

With...End With, instruction (Visual Basic)

Concepts

Structures de décision

Structures de boucle

Autres structures de contrôle

Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Structures de contrôle imbriquées

Vous pouvez insérer des instructions de contrôle dans d'autres instructions de contrôle, par exemple un bloc **If...Then...Else** dans une boucle **For...Next**. Dans ce cas, l'instruction de contrôle est *imbriquée* dans une autre instruction de contrôle.

Niveaux d'imbrication

Dans Visual Basic, les structures de contrôle peuvent être imbriquées sur autant de niveaux que nécessaire. Il est courant de mettre en retrait le corps de chaque structure imbriquée afin d'accroître la lisibilité. L'éditeur d'environnement de développement intégré le fait automatiquement.

Dans l'exemple suivant, la procédure `sumRows` additionne les éléments positifs de chaque ligne de la matrice.

```
Public Sub sumRows(ByVal a(,) As Double, ByRef r() As Double)
    Dim i, j As Integer
    For i = 0 To UBound(a, 1)
        r(i) = 0
        For j = 0 To UBound(a, 2)
            If a(i, j) > 0 Then
                r(i) = r(i) + a(i, j)
            End If
        Next j
    Next i
End Sub
```

Dans l'exemple précédent, la première instruction **Next** ferme la boucle **For** interne et la dernière instruction **Next** ferme la boucle **For** externe.

De même, dans les instructions **If** imbriquées, les instructions **End If** s'appliquent automatiquement à l'instruction **If** précédente la plus proche. Les boucles **Do** imbriquées fonctionnent de la même manière, dans le sens où l'instruction **Loop** la plus profonde correspond à l'instruction **Do** la plus profonde.

Imbrication de genres de structures de contrôle différents

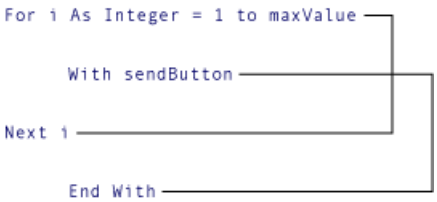
Vous pouvez imbriquer un genre de structure de contrôle dans un autre. L'exemple suivant utilise un bloc **With** à l'intérieur d'une boucle **For Each** et des blocs **If** imbriqués à l'intérieur du bloc **With**.

```
For Each ctl As System.Windows.Forms.Control In Me.Controls
    With ctl
        .BackColor = System.Drawing.Color.Yellow
        .ForeColor = System.Drawing.Color.Black
        If .CanFocus Then
            .Text = "Colors changed"
            If Not .Focus() Then
                ' Insert code to process failed focus.
            End If
        End If
    End With
Next ctl
```

Chevauchement de structures de contrôle

Vous ne devez pas faire en sorte que des structures de contrôle se chevauchent. Cela signifie que toute structure imbriquée doit être entièrement contenue dans la structure la plus profonde suivante. Par exemple, la disposition suivante est non valide car la boucle **For** se termine avant la fin du bloc **With** interne.

Imbrication non valide de structures For et With



Le compilateur Visual Basic détecte ces chevauchements de structures de contrôle et signale une erreur de compilation.

Voir aussi

Tâches

Comment : transférer un contrôle hors d'une structure de contrôle

Concepts

Structures de décision

Structures de boucle

Autres structures de contrôle

Autres ressources

Flux de contrôle en Visual Basic

Comment : transférer un contrôle hors d'une structure de contrôle

Vous pouvez utiliser l'instruction Exit, instruction (Visual Basic) pour quitter directement une structure de contrôle. **Exit** transfère immédiatement l'exécution à l'instruction qui suit la dernière instruction de la structure de contrôle. La syntaxe de l'instruction **Exit** spécifie le type de structure de contrôle que vous quittez. Les versions suivantes de l'instruction **Exit** sont possibles :

- **Exit Select**
- **Exit Try**
- **Exit While**
- **Exit Do**
- **Exit For**

Les instructions **Exit** peuvent apparaître autant de fois que nécessaire dans une structure de contrôle qui les prend en charge. **Exit** est utile lorsqu'une structure de contrôle a fait tout ce qu'elle avait à faire et n'a plus aucune instruction à exécuter.

Structures de contrôle qui ne prennent pas en charge Exit. Vous ne pouvez pas utiliser l'instruction **Exit** pour quitter un bloc **If**, **Using** ou **With**. Pour obtenir le même résultat, vous pouvez placer une étiquette d'instruction sur l'instruction **End** du bloc et y accéder à l'aide d'une instruction **GoTo**. Pour plus d'informations sur les étiquettes d'instructions, consultez Comment : étiqueter des instructions.

Exemple

Si une instruction **Exit** est rencontrée dans des structures de contrôle imbriquées, le contrôle passe à l'instruction qui suit la fin de la structure la plus profonde du type spécifié dans l'instruction **Exit**. L'exemple suivant illustre ce comportement :

```
Public Sub invertElements(ByRef a(,) As Double)
    For i As Integer = 0 To UBound(a, 1)
        For j As Integer = 0 To UBound(a, 2)
            If a(i, j) = 0 Then
                ' Cannot complete this row; resume outer loop.
                Exit For
            Else
                a(i, j) = 1.0 / a(i, j)
            End If
        Next j
        ' Control comes here directly from the Exit For statement.
    Next i
End Sub
```

Dans l'exemple précédent, l'instruction **Exit For** est située dans la boucle **For** interne ; elle passe donc le contrôle à l'instruction qui suit la boucle et poursuit avec la boucle **For** externe.

Voir aussi

Tâches

Comment : étiqueter des instructions

Référence

Exit, instruction (Visual Basic)

GoTo, instruction

Concepts

Structures de décision

Structures de boucle

Autres structures de contrôle

Structures de contrôle imbriquées

Autres ressources

Flux de contrôle en Visual Basic

Programmation orientée objet dans Visual Basic

Les objets sont au centre de la programmation Visual Basic : les formulaires et les contrôles sont des objets. ainsi que les bases de données. S'il vous est arrivé d'utiliser Visual Basic ou si vous avez utilisé les exemples fournis dans la documentation, vous avez déjà programmé des objets. Mais il existe d'autres utilisations des objets différentes de celles que vous avez vues jusqu'ici.

Dans les rubriques suivantes, vous pourrez constater qu'il est facile de créer vos propres objets depuis les classes que vous définissez. Vous verrez également comment utiliser des objets pour simplifier le codage et augmenter la réutilisation du code.

Dans cette section

Introduction aux objets dans Visual Basic

Fournit une introduction aux termes et aux concepts utilisés dans la programmation orientée objet.

Liaison anticipée et liaison tardive

Décrit la liaison effectuée par le compilateur lorsqu'un objet est assigné à une variable objet et les différences entre des objets à liaison anticipée et à liaison tardive.

Membres partagés en Visual Basic

Décrit les membres pouvant être utilisés sans instances.

Création et utilisation d'objets

Montre comment créer et utiliser des instances de classes.

Gestion de groupes d'objets

Fournit des techniques de travail sur les tableaux et les collections d'objets

Obtention d'informations sur la classe au moment de l'exécution

Explique comment déterminer la classe à laquelle appartient un objet.

Présentation des classes

Présente des instructions détaillées sur la création d'objet et les problèmes de durée de vie.

Événements en Visual Basic

Montre comment déclarer et utiliser des événements.

Délégués en Visual Basic

Montre comment déclarer et utiliser des délégués.

Interfaces en Visual Basic

Définit les interfaces et présente leur utilisation dans vos applications.

Héritage en Visual Basic

Explique comment définir des classes servant de base à d'autres classes.

Rubriques connexes

Objets en Visual Basic

Définit les objets et présente leur utilisation.

Programmation à l'aide de composants

Explique comment utiliser Visual Basic pour contrôler des objets fournis par d'autres applications.

Introduction aux objets dans Visual Basic

Un *objet* est une structure contenant des données et des méthodes qui manipulent les données. La plupart des opérations effectuées dans Visual Basic implique des objets. Si vous n'êtes pas familiarisé avec la programmation orientée objet, les termes et les concepts suivants vont vous aider à démarrer.

Classes et objets

Les termes « classe » et « objet » sont tellement utilisés dans la programmation orientée objet qu'ils peuvent être facilement confondus. En règle générale, une *class* est une représentation abstraite de quelque chose, tandis qu'un *objet* est un exemple utilisable de ce que représente la classe. La seule exception à cette règle concerne les membres de classe partagés, qui peuvent être utilisés dans les instances d'une classe et dans les variables d'objet déclarées comme le type de la classe.

Champs, propriétés, méthodes et événements

Les classes se composent de champs, de propriétés, de méthodes et d'événements. Les propriétés et les champs sont des informations contenues dans un objet. Les champs sont similaires aux variables, car ils peuvent être lus ou définis directement. Par exemple, si vous possédez un objet intitulé "Car", vous pouvez stocker sa couleur dans un champ nommé "Color".

Les propriétés sont extraites et définies comme des champs, mais sont implémentées via les procédures de propriété **Get** et **Set** qui contrôlent davantage la façon dont les valeurs sont définies ou retournées. La couche d'adressage entre la valeur stockée et les procédures qui utilisent cette valeur vous aide à isoler vos données et vous permet de valider les valeurs avant qu'elles ne soient attribuées ou extraites.

Les méthodes représentent des actions que peut effectuer un objet. Par exemple, un objet "Car" peut avoir des méthodes "StartEngine", "Drive" et "Stop". Pour définir des méthodes, ajoutez des procédures (des fonctions ou des routines **Sub**) à votre classe.

Les événements vous informent lorsqu'un objet est reçu d'autres applications ou transmis à celles-ci. Les événements permettent aux objets d'effectuer des actions lorsqu'une occurrence spécifique se produit. Un exemple d'un événement de la classe "Car" est un événement "Check_Engine". Dans la mesure où Microsoft Windows est un système d'exploitation piloté par événements, les événements peuvent provenir d'autres objets, applications ou actions de l'utilisateur (par exemple, lorsqu'il clique sur la souris ou appuie sur une touche).

Encapsulation, héritage et polymorphisme

Les champs, les propriétés, les méthodes et les événements ne représentent que la moitié de l'équation de la programmation orientée objet. La programmation orientée objet réelle nécessite que les objets prennent en charge trois qualités : l'encapsulation, l'héritage et le polymorphisme.

L'*encapsulation* signifie qu'un groupe de propriétés, méthodes et autres membres liés est traité comme une unité ou un objet unique. Les objets peuvent contrôler la façon dont les propriétés sont modifiées ou la façon dont les méthodes sont exécutées. Par exemple, un objet peut valider des valeurs avant d'autoriser des modifications de propriété. L'encapsulation facilite également la modification ultérieure de votre implémentation en vous permettant de masquer les détails d'implémentation de vos objets ; cette méthode est connue sous le nom de *masquage des données*.

L'*héritage* décrit la possibilité de créer de nouvelles classes en fonction d'une classe existante. La nouvelle classe hérite de toutes les propriétés, des méthodes et des événements de la classe de base et peut être personnalisée avec des propriétés et des méthodes supplémentaires. Par exemple, vous pouvez créer une classe intitulée "Truck" basée sur la classe "Car". La classe "Truck" hérite de la propriété "Color" de la classe "Car" et peut comporter des propriétés supplémentaires, telles que "FourWheelDrive".

Le *polymorphisme* signifie que plusieurs classes peuvent être utilisées de manière interchangeable, même si chacune des classes implémente les mêmes propriétés et méthodes de manière différente. Le polymorphisme est primordial dans la programmation orientée objet car il vous permet d'utiliser des éléments portant le même nom, quel que soit le type d'objet utilisé actuellement. Pour une classe de base "Car", par exemple, le polymorphisme permet au programmeur de définir plusieurs méthodes "StartEngine" pour tout nombre de classes dérivées. La méthode "StartEngine" d'une classe dérivée appelée "DieselCar" peut être totalement différente de la méthode portant le même nom dans la classe de base. D'autres procédures ou méthodes peuvent utiliser la méthode "StartEngine" des classes dérivées exactement de la même façon, quel que soit le type d'objet "Car" utilisé simultanément.

Surcharge, substitution et occultation

La surcharge, la substitution et l'occultation sont des concepts similaires pouvant être facilement confondus. Même si ces trois

techniques vous permettent de créer des membres portant le même nom, elles présentent d'importantes différences.

- Les membres surchargés sont utilisés pour fournir des versions différentes d'une propriété ou d'une méthode portant le même nom, mais qui acceptent divers nombres ou paramètres ou des paramètres de divers types de données.
- Les propriétés ou les méthodes substituées permettent de remplacer une propriété ou une méthode héritée qui n'est pas adaptée dans une classe dérivée. Les membres substitués doivent accepter le même type de données ainsi que le même nombre d'arguments. Les classes dérivées héritent des membres substitués.
- Les membres occultés permettent de remplacer localement un membre dont la portée est plus vaste. Tout type peut occulter un autre type. Par exemple, vous pouvez déclarer une propriété qui occulte une méthode héritée du même nom. Les membres occultés ne peuvent pas être hérités.

Voir aussi

Concepts

Classes : modèles d'objets
Propriétés et méthodes surchargées
Substitution de propriétés et de méthodes
Occultation dans Visual Basic

Autres ressources

Objets en Visual Basic
Création et utilisation d'objets
Héritage en Visual Basic

Présentation des classes

Les classes revêtent de l'importance dans la programmation orientée objet car elles permettent de regrouper des éléments en une seule entité, mais aussi d'en contrôler la visibilité et l'accessibilité par d'autres procédures. En outre, une classe peut hériter du code défini dans d'autres classes et le réutiliser.

Dans cette section

Classes : modèles d'objets

Propose une vue d'ensemble de l'encapsulation, de l'héritage et des membres partagés.

Procédure pas à pas : définition de classes

Décrit étape par étape le processus de création d'une classe.

Classes et modules

Explique les différences qui distinguent les classes des modules standard.

Durée de vie d'un objet : création et destruction des objets

Analyse la création et la suppression des instances de classe.

Comment : implémenter le modèle DisposeFinalize (Visual Basic)

Explique comment vérifier que les ressources d'un objet sont libérées lorsqu'elles deviennent inutiles.

Utilisation de constructeurs et de destructeurs

Traite de l'initialisation de nouvelles instances de classes et de la suppression des ressources devenues inutiles.

Propriétés, champs et méthodes de classe

Décrit les champs, les propriétés et les méthodes qui constituent une classe.

Rubriques connexes

Création et utilisation d'objets

Illustre la création et l'utilisation d'instances de classes.

Interfaces en Visual Basic

Décrit ce que représentent les interfaces et comment les utiliser dans vos applications.

Héritage en Visual Basic

Explique comment définir des classes servant de base à d'autres classes.

Classes : modèles d'objets

Les classes sont des représentations symboliques d'objets ; elles décrivent les propriétés, les champs, les méthodes et les événements qui constituent des objets de la même façon que les plans sont des modèles décrivant les éléments qui composent un bâtiment. Tout comme un plan peut être employé pour construire plusieurs bâtiments, une classe unique peut servir à créer autant d'objets que cela est nécessaire. Et tout comme un plan définit les parties d'un bâtiment qui sont accessibles aux personnes utilisant celui-ci, les classes peuvent elles aussi contrôler l'accès des utilisateurs aux composants des objets par l'encapsulation.

Classes et objets

Les termes *classe* et *objet* sont parfois employés indifféremment, mais en réalité, les classes décrivent la *structure* des objets, alors que ces derniers sont des *instances* utilisables des classes. Chaque instance est une copie exacte, mais distincte de sa classe. Dans la mesure où un objet est une instance d'une classe, l'action de créer un objet se nomme *instanciation*.

Si l'on reprend l'analogie avec un plan de construction, une classe est un plan, et un objet est un bâtiment construit à partir de ce plan. Habituellement, la modification des données dans un objet ne modifie les données dans aucun autre objet. (les *membres partagés* qui sont des membres de classe déclarés avec le modificateur **Shared**, font toutefois exception à cette règle puisqu'ils existent indépendamment des instances spécifiques d'une classe).

Encapsulation

L'*encapsulation* est la capacité à contenir un groupe d'éléments associés et à en contrôler l'accès. Les classes constituent l'un des moyens les plus courants d'encapsuler des éléments. Dans l'exemple ci-dessous, la classe `BankAccount` encapsule les méthodes, les champs et les propriétés qui décrivent un compte bancaire.

Sans encapsulation, il aurait été nécessaire de déclarer des procédures et des variables distinctes pour stocker et gérer les informations relatives au compte bancaire, sans parler de la difficulté de manipuler plus d'un seul compte bancaire à la fois. L'encapsulation vous permet d'utiliser les données et les procédures de la classe `BankAccount` en tant qu'unité. Vous pouvez manipuler plusieurs comptes bancaires à la fois sans qu'il y ait confusion, car chaque compte est représenté par une instance unique de la classe.

L'encapsulation permet en outre de contrôler la manière dont les données et les procédures sont utilisées. Vous pouvez recourir à des modificateurs d'accès, tels que **Private** ou **Protected**, pour empêcher les procédures externes d'exécuter des méthodes de classe ou de lire et de modifier les données figurant dans les propriétés et les champs. Vous devez déclarer les détails internes d'une classe en tant que **Private** pour éviter qu'ils soient utilisés en dehors de votre classe ; cette technique est appelée le *masquage de données* et permet de protéger des informations client, telles que le solde d'un compte.

L'une des règles fondamentales de l'encapsulation est que des données de classe doivent être modifiées ou extraites uniquement par des méthodes ou des procédures **Property**. Le fait de masquer les informations d'implémentation des classes vous permet d'empêcher que celles-ci ne soient utilisées de façon non souhaitée et de modifier ces éléments ultérieurement sans risquer des problèmes de compatibilité. Par exemple, les versions ultérieures de la classe `BankAccount` peuvent modifier le type de données du champ `AccountBalance` sans entraver le fonctionnement des applications pour lesquelles ce champ doit absolument présenter un type de données spécifique.

Héritage

Comme les structures Visual Basic, vous pouvez utiliser des classes pour définir des types de données qui encapsulent un groupe d'éléments connexes. Contrairement aux structures, toutefois, les classes Visual Basic peuvent hériter des caractéristiques d'autres classes et étendre ces caractéristiques. Les classes qui servent de base à de nouvelles classes sont appelées *classes de base*. Les classes dérivées des classes de base sont appelées *classes dérivées*. Les classes dérivées héritent de tous les champs, propriétés, méthodes et événements de leur classe de base. Cela signifie que vous pouvez développer et déboguer une classe une seule fois et la réutiliser ensuite comme la classe de base d'autres classes.

L'exemple suivant définit une classe de base qui représente un compte bancaire générique, et une classe spécifique qui hérite des propriétés de la classe de base, mais qui est personnalisée pour décrire un compte de chèques.

VB

```
Class BankAccount
    Private AccountNumber As String
    Private AccountBalance As Decimal
    Private HoldOnAccount As Boolean = False
    Public Sub PostInterest()
        ' Add code to calculate the interest for this account.
```

```
    End Sub
    ReadOnly Property Balance() As Decimal
        Get
            ' Return the available balance.
            Return AccountBalance
        End Get
    End Property
End Class

Class CheckingAccount
    Inherits BankAccount
    Sub ProcessCheck()
        ' Add code to process a check drawn on this account.
    End Sub
End Class
```

Pour plus d'informations sur l'héritage, consultez Éléments fondamentaux de l'héritage.

Membres partagés

Par défaut, les données de classe sont spécifiques à chaque instance de la classe, mais il peut arriver qu'une donnée unique doive être partagée entre tous les objets créés à partir d'une classe. Dans de tels cas, utilisez le modificateur **Shared** pour qu'une variable partage la même valeur dans toutes les instances d'une classe (les membres Shared sont parfois désignés sous le nom de « membres statiques » dans d'autres langages de programmation). Vous pouvez appeler des méthodes partagées directement à l'aide d'un nom de classe sans créer au préalable une instance de la classe.

Pour plus d'informations sur les membres partagés, consultez Membres partagés en Visual Basic.

Occultation

Les classes dérivées peuvent utiliser le mot clé **Shadows** pour déclarer un membre avec le même nom que le membre hérité. Les membres occultés ne doivent pas forcément être du même type de données que le membre à occulter. Par exemple, une propriété peut occulter une variable de type **Integer**.

Pour plus d'informations sur les membres partagés, consultez Occultation dans Visual Basic.

Voir aussi

Référence

Shadows
Shared (Visual Basic)

Concepts

Structures et classes
Éléments fondamentaux de l'héritage
Membres partagés en Visual Basic
Occultation dans Visual Basic

Autres ressources

Création et utilisation d'objets
Héritage en Visual Basic

Procédure pas à pas : définition de classes

Cette procédure pas à pas montre comment définir des classes que vous pouvez ensuite utiliser pour créer des objets. Elle montre également comment ajouter des propriétés et des méthodes à la nouvelle classe et illustre l'initialisation d'un objet.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour définir une classe

1. Créez un projet en cliquant sur **Nouveau projet** dans le menu **Fichier**. La boîte de dialogue **Nouveau projet** s'affiche.
2. Sélectionnez Application Windows dans la liste des modèles de projet Visual Basic pour afficher le nouveau projet.
3. Ajoutez une nouvelle classe au projet en cliquant sur **Ajouter une classe** dans le menu **Projet**. La boîte de dialogue **Ajouter un nouvel élément** s'affiche alors.
4. Sélectionnez le modèle **Classe**.
5. Nommez la nouvelle classe `UserNameInfo.vb`, puis cliquez sur **Ajouter** pour afficher le code de la nouvelle classe.

   VB

   ```
   Public Class UserNameInfo
   End Class
   ```

   Remarque

   Vous pouvez utiliser l'**éditeur de code** de Visual Basic pour ajouter une classe à votre formulaire de démarrage en tap ant le mot clé **Class**, suivi du nom de la nouvelle classe. L'**éditeur de code** fournit une instruction **End Class** correspo ndante.

6. Définissez un champ privé pour la classe en insérant le code suivant entre les instructions **Class** et **End Class** :

   VB

   ```
   Private userNameValue As String
   ```

   La déclaration du champ comme **Private** signifie qu'il peut être utilisé uniquement dans la classe. Vous pouvez rendre les champs accessibles à partir de l'extérieur d'une classe en utilisant des modificateurs d'accès qui fournissent un accès plus grand, tel que l'accès **Public**. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

7. Définissez une propriété pour la classe en ajoutant le code suivant :

   VB

   ```
   Public Property UserName() As String
       Get
           ' Gets the property value.
           Return userNameValue
       End Get
       Set(ByVal Value As String)
           ' Sets the property value.
           userNameValue = Value
       End Set
   End Property
   ```

8. Définissez une méthode pour la classe en ajoutant le code suivant :

   VB

```
Public Sub Capitalize()
    ' Capitalize the value of the property.
    userNameValue = UCase(userNameValue)
End Sub
```

9. Définissez un constructeur paramétré pour la nouvelle classe en ajoutant une procédure nommée `Sub New` :

   VB

```
Public Sub New(ByVal UserName As String)
    ' Set the property value.
    Me.UserName = UserName
End Sub
```

   Le constructeur `Sub New` est appelé automatiquement lorsqu'un objet basé sur cette classe est créé. Ce constructeur définit la valeur du champ qui contient le nom d'utilisateur.

Pour créer un bouton permettant de tester la classe

1. Modifiez le formulaire de démarrage en mode design en cliquant avec le bouton droit sur son nom dans l'**Explorateur de solutions**, puis en cliquant sur **Concepteur de vues**. Par défaut, le formulaire de démarrage des projets d'application Windows est intitulé Form1.vb. Le formulaire principal apparaît ensuite.
2. Ajoutez un bouton au formulaire principal et double-cliquez dessus pour afficher le code du gestionnaire d'événements `Button1_Click`. Ajoutez le code suivant pour appeler la procédure de test :

   VB

```
' Create an instance of the class.
Dim user As New UserNameInfo("Moore, Bobby")
' Capitalize the value of the property.
user.Capitalize()
' Display the value of the property.
MsgBox("The original UserName is: " & user.UserName)
' Change the value of the property.
user.UserName = "Worden, Joe"
' Redisplay the value of the property.
MsgBox("The new UserName is: " & user.UserName)
```

Pour exécuter l'application

1. Exécutez l'application en appuyant sur F5. Cliquez sur le bouton situé sur votre formulaire pour appeler la procédure de test. Affiche un message indiquant que le `UserName` d'origine est "MOORE, BOBBY", parce que la procédure a appelé la méthode `Capitalize` de l'objet.
2. Cliquez sur **OK** pour fermer le message. La procédure `Button1 Click` modifie la valeur de la propriété `UserName` et affiche un message qui indique que la nouvelle valeur de `UserName` est "Worden, Joe".

Voir aussi

Tâches

Comment : ajouter des événements à une classe

Référence

Imports, instruction

Concepts

Classes et modules

Autres ressources

Présentation des classes

Propriétés, champs et méthodes de classe

Classes et modules

Les classes et les modules sont des types référence qui encapsulent des éléments définis en leur sein, mais ils se distinguent par la manière dont ces éléments sont accessibles à partir d'autres procédures.

Différences entre les classes et les modules

La principale différence qu'il existe entre les classes et les modules est que les classes peuvent être instanciées comme des objets, mais pas les modules standard. Comme il n'existe qu'une seule copie des données d'un module standard, quand une partie de votre programme transforme une variable publique en module standard, toute autre partie du programme reçoit la même valeur si, par la suite, elle lit cette variable. Par opposition, les données d'objet existent séparément pour chaque objet instancié. Une autre différence est que les classes peuvent implémenter des interfaces, contrairement aux modules standard.

Remarque

Lorsque le modificateur **Shared** est appliqué à un membre de classe, il est associé à la classe elle-même au lieu d'une instance particulière de la classe. Le membre est accessible directement via le nom de classe au même titre que les membres de module. Pour plus d'informations sur les membres partagés, consultez Membres partagés en Visual Basic.

Les classes et les modules utilisent également des portées différentes pour leurs membres. Les membres définis dans une classe ont comme portée une instance spécifique de la classe et n'existent que pendant la durée de vie de l'objet. Pour accéder aux membres de classe depuis l'extérieur d'une classe, vous devez utiliser des noms qualifiés complets au format *Object.Member*.

En revanche, les membres déclarés dans un module sont publiquement accessibles par défaut et sont accessibles par tout code qui peut accéder au module. Dès lors, les variables d'un module standard sont effectivement des variables globales puisqu'elles sont visibles à n'importe quel emplacement du projet et qu'elles existent pendant toute la durée de vie du programme.

Voir aussi

Référence

Shared (Visual Basic)

Concepts

Structures et classes

Membres partagés en Visual Basic

Autres ressources

Présentation des classes

Durée de vie d'un objet : création et destruction des objets

Une instance d'une classe, d'un objet est créée à l'aide du mot clé **New**. Des tâches d'initialisation doivent souvent être effectuées sur les nouveaux objets avant qu'ils ne soient utilisés. Les tâches d'initialisation courantes comprennent l'ouverture de fichiers, la connexion aux bases de données et la lecture des valeurs de clés de Registre. Microsoft Visual Basic 2005 contrôle l'initialisation des nouveaux objets à l'aide de procédures appelées *constructeurs* (méthodes spéciales permettant de contrôler l'initialisation).

Lorsqu'un objet abandonne la portée, il est libéré par le Common Language Runtime (CLR). Visual Basic 2005 contrôle la libération des ressources système à l'aide de procédures nommées *destructeurs*. Les constructeurs et les destructeurs prennent en charge ensemble la création de bibliothèques de classe robustes et prévisibles.

Sub New et Sub Finalize

Les procédures **Sub New** et **Sub Finalize** de Visual Basic 2005 initialisent et détruisent des objets ; ils remplacent les méthodes **Class_Initialize** et **Class_Terminate** utilisées dans Visual Basic 6.0 et les versions précédentes. Contrairement à **Class_Initialize**, le constructeur **Sub New** ne peut s'exécuter qu'après la création d'une classe. Il ne peut être appelé explicitement qu'à partir de la première ligne de code d'un autre constructeur de la même classe ou d'une classe dérivée. En outre, le code de la méthode **Sub New** s'exécute toujours avant tout autre code dans une classe. Visual Basic 2005 crée implicitement un constructeur **Sub New** au moment de l'exécution si vous ne définissez pas explicitement de procédure **Sub New** pour une classe.

Avant de libérer des objets, le CLR appelle automatiquement la méthode **Finalize** pour les objets qui définissent une procédure **Sub Finalize**. La méthode **Finalize** peut contenir du code qui doit s'exécuter juste avant la destruction d'un objet, par exemple du code pour la fermeture de fichiers et l'enregistrement d'informations d'état. L'exécution de la méthode **Sub Finalize** réduit légèrement les performances, c'est pourquoi vous ne devez exécuter une méthode **Sub Finalize** que si vous devez libérer des objets explicitement.

Remarque
Le garbage collector du CLR ne peut pas supprimer les *objets non managés*, objets que le système d'exploitation exécute dire ctement, en dehors de l'environnement du CLR. Il en est ainsi parce que les différents objets non managés doivent être suppr imés de différentes façons. Ces informations ne sont pas directement associées à l'objet non managé ; elles doivent être rech erchées dans la documentation de l'objet. Une classe qui utilise des objets non managés doit les supprimer dans sa méthode **Finalize**.

Le destructeur **Finalize** est une méthode protégée qui peut être appelée uniquement à partir de la classe à laquelle elle appartient ou de classes dérivées. Le système appelle automatiquement **Finalize** lors de la destruction d'un objet, c'est pourquoi il ne faut pas appeler explicitement **Finalize** depuis l'extérieur de l'implémentation **Finalize** d'une classe dérivée.

Contrairement à **Class_Terminate** qui s'exécute dès qu'un objet a une valeur nulle, il y a généralement un délai entre le moment ou un objet perd la portée et le moment où Visual Basic 2005 appelle le destructeur **Finalize**. Visual Basic 2005 autorise un deuxième type de destructeur, Dispose, qui peut être explicitement appelé à n'importe quel moment pour libérer immédiatement des ressources.

Remarque
Un destructeur **Finalize** ne doit pas lever d'exceptions, parce qu'elles ne peuvent pas être gérées par l'application et peuvent provoquer l'arrêt de cette dernière.

Interface IDisposable

Les instances de classe contrôlent souvent les ressources non managées par le CLR, comme les handles Windows et les connexions aux bases de données. Ces ressources doivent être supprimées dans la méthode **Finalize** de la classe, afin qu'elles soient libérées lorsque l'objet est détruit par le garbage collector. Toutefois, le garbage collector ne détruit les objets que si le CLR requiert plus de mémoire disponible. Cela signifie que les ressources ne peuvent pas être libérées tant que l'objet n'abandonne pas la portée.

Pour compléter le garbage collection, vos classes peuvent fournir un mécanisme permettant de gérer activement les ressources système si elles implémentent l'interface IDisposable. **IDisposable** a une méthode, **Dispose**, que les clients doivent appeler lorsqu'ils ont fini d'utiliser un objet. Vous pouvez employer la méthode **Dispose** pour libérer immédiatement des ressources et effectuer des tâches comme la fermeture de fichiers et les connexions à une base de données. Contrairement au

destructeur **Finalize**, la méthode **Dispose** n'est pas appelée automatiquement. Les clients d'une classe doivent appeler explicitement **Dispose** pour pouvoir libérer immédiatement des ressources.

Implémentation de l'interface IDisposable

Une classe qui implémente l'interface **IDisposable** doit inclure les sections de code ci-dessous :

- un champ permettant de vérifier la suppression ou non de l'objet :

```
Protected disposed As Boolean = False
```

- surcharge du **Dispose** qui libère les ressources de la classe. Cette méthode doit être appelée par les méthodes **Dispose** et **Finalize** de la classe de base :

```
Protected Overridable Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposed Then
        If disposing Then
            ' Insert code to free unmanaged resources.
        End If
        ' Insert code to free shared resources.
    End If
    Me.disposed = True
End Sub
```

- une implémentation de **Dispose** qui contient uniquement le code suivant :

```
Public Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
    GC.SuppressFinalize(Me)
End Sub
```

- une substitution de la méthode **Finalize** qui contient uniquement le code suivant :

```
Protected Overrides Sub Finalize()
    Dispose(False)
    MyBase.Finalize()
End Sub
```

Dérivation d'une classe qui implémente IDisposable

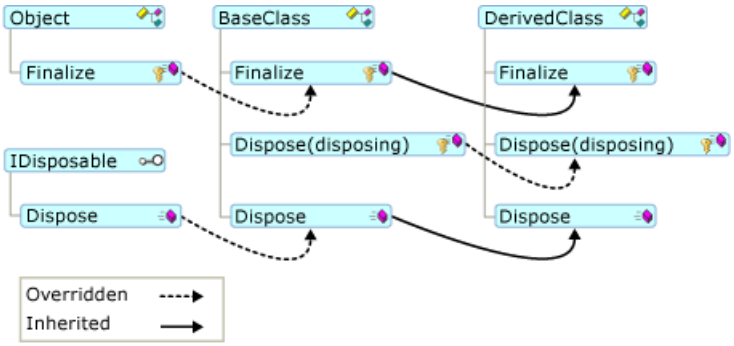
Une classe qui dérive d'une classe de base implémentant l'interface **IDisposable** n'a pas besoin de substituer chacune des méthodes de base à moins qu'elle n'utilise des ressources supplémentaires qui doivent être supprimées. Dans cette situation, la classe dérivée doit substituer la méthode `Dispose(disposing)` de la classe de base pour supprimer les ressources de la classe dérivée. Cette substitution doit appeler la méthode `Dispose(disposing)` de la classe de base.

```
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposed Then
        If disposing Then
            ' Insert code to free unmanaged resources.
        End If
        ' Insert code to free shared resources.
    End If
    MyBase.Dispose(disposing)
End Sub
```

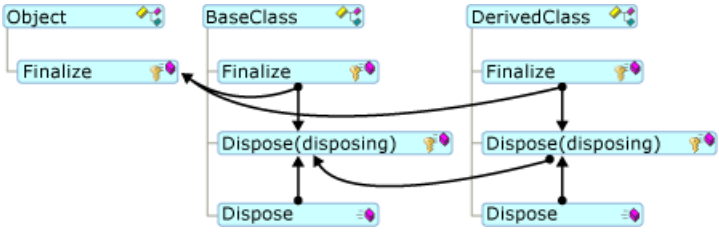
Une classe dérivée ne doit pas substituer les méthodes **Dispose** et **Finalize** de la classe de base. Lorsque ces méthodes sont appelées à partir d'une instance de la classe dérivée, l'implémentation de la classe de base de ces méthodes appelle la substitution de la classe dérivée de la méthode `Dispose(disposing)`.

Visualisation

Le diagramme suivant affiche les méthodes qui sont héritées et celles qui sont substituées dans la classe dérivée.



Lorsque ce modèle **Dispose Finalize** est suivi, les ressources de la classe dérivée et de la classe de base sont supprimées correctement. Le diagramme suivant affiche les méthodes qui sont appelées lorsque les classes sont supprimées et finalisées.



Garbage collection et destructeur Finalize

Le .NET Framework utilise le système *garbage collection par traçage des références* pour libérer régulièrement les ressources inutiles. Visual Basic 6.0 et les versions antérieures géraient les ressources à l'aide d'un système différent, appelé *décompte de références*. Bien que ces deux systèmes exécutent la même fonction automatiquement, un certain nombre de différences importantes les distinguent.

Le CLR détruit régulièrement les objets quand le système détermine que ces objets sont devenus inutiles. Les objets sont libérés plus vite en cas de pénurie des ressources système et moins vite dans le cas contraire. Le délai qui sépare le moment où la portée d'un objet se termine et le moment où le CLR libère celui-ci implique qu'il est désormais impossible de déterminer avec précision l'instant où l'objet sera détruit, alors que cela était possible en Visual Basic 6.0 et les versions précédentes. Dans une telle situation, l'on dit que les objets ont une *durée de vie non déterminable*. Dans la plupart des cas, la durée de vie non déterminable ne modifie en rien la manière dont vous écrivez les applications, à condition de ne pas oublier que le destructeur **Finalize** ne s'exécutera peut-être pas immédiatement au moment où la portée d'un objet s'achève.

Une autre différence entre les deux systèmes d'opérations garbage collection concerne l'utilisation de **Nothing**. Pour tirer profit du décompte de références dans Visual Basic 6.0 et les versions précédentes, les programmeurs ont parfois assigné **Nothing** aux variables objets pour libérer les références contenues dans ces variables. Si la variable contenait la dernière référence à l'objet, les ressources de ce dernier ont été libérées immédiatement. Dans Visual Basic 2005, si l'emploi de cette procédure présente sans doute toujours un intérêt dans certaines situations, il n'entraîne jamais la libération immédiate des ressources de l'objet référencé. Pour libérer immédiatement des ressources, utilisez la méthode **Dispose** de l'objet, si disponible. La seule situation qui nécessite l'attribution de **Nothing** à une variable est lorsque cette variable possède une durée de vie qui est longue par rapport à la période requise pour la détection des objets orphelins par le garbage collector.

Voir aussi

Tâches

Comment : implémenter le modèle DisposeFinalize (Visual Basic)

Référence

Utilisation de constructeurs et de destructeurs
New (Visual Basic)
Nothing (Visual Basic)
Dispose

Concepts

Initialisation et suppression des composants
Destructeurs et méthodes Finalize

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Comment : implémenter le modèle DisposeFinalize (Visual Basic)

Le modèle **Dispose Finalize** vérifie que les ressources sont libérées lorsque l'objet n'est plus nécessaire.

Exemple

La classe `ResourceClass` dans l'exemple suivant utilise des ressources managées et non managées, puis utilise le modèle **Dispose Finalize** pour les supprimer correctement. Les ressources et leurs fonctions sont les suivantes :

- L'implémentation de la méthode Dispose autorise les utilisateurs de la classe à supprimer des instances de classe. Cette méthode appelle `Dispose(True)` pour supprimer les ressources de l'objet, puis appelle SuppressFinalize pour empêcher le code de finalisation de s'exécuter une deuxième fois.
- La substitution de la méthode Finalize de base autorise le garbage collector du Common Language Runtime (CLR) à supprimer des instances de classe. Cette méthode appelle `Dispose(False)` pour supprimer les ressources de l'objet. Notez que si **Dispose** a été appelé précédemment pour l'objet, son appel à **SuppressFinalize** empêche le garbage collector d'appeler la méthode **Finalize**.
- La surcharge de la méthode **Dispose** exécute le travail de suppression. Elle accepte un paramètre Boolean, `disposing`, qui indique si votre code a initialisé la suppression de l'objet. Lorsque vous supprimez un objet, toutes ses ressources doivent être supprimées. Lorsque le garbage collector de l'environnement CLR supprime un objet, seules les ressources non managées doivent être supprimées ; le garbage collector supprime automatiquement les ressources managées, si nécessaire.

Pour plus d'informations, consultez Durée de vie d'un objet : création et destruction des objets.

VB

```
    Public Class ResourceClass
        Implements IDisposable

        Private managedResource As System.ComponentModel.Component
        Private unmanagedResource As IntPtr
        Protected disposed As Boolean = False

        Public Sub New()
            ' Insert appropriate constructor code here.
        End Sub

        Protected Overridable Overloads Sub Dispose( _
            ByVal disposing As Boolean)
            If Not Me.disposed Then
                If disposing Then
                    managedResource.Dispose()
                End If
                ' Add code here to release the unmanaged resource.
                unmanagedResource = IntPtr.Zero
                ' Note that this is not thread safe.
            End If
            Me.disposed = True
        End Sub

        Public Sub AnyOtherMethods()
            If Me.disposed Then
                Throw New ObjectDisposedException(Me.GetType().ToString, _
                    "This object has been disposed.")
            End If
        End Sub

#Region " IDisposable Support "
        ' Do not change or add Overridable to these methods.
        ' Put cleanup code in Dispose(ByVal disposing As Boolean).
        Public Overloads Sub Dispose() Implements IDisposable.Dispose
```

```
            Dispose(True)
            GC.SuppressFinalize(Me)
        End Sub
        Protected Overrides Sub Finalize()
            Dispose(False)
            MyBase.Finalize()
        End Sub
#End Region
    End Class
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve dans **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite les éléments suivants :

- l'accès aux membres des espaces de noms System et System.ComponentModel. Ajoutez une instruction **Imports** si vous n'utilisez pas des noms de membres qualifiés complets dans votre code. Pour plus d'informations, consultez Imports, instruction.

Ces modifications doivent être effectuées dans le code :

- Remplacez `ResourceClass` par le nom de la classe qui implémente **IDisposable**.
- Utilisez le test de `AnyOtherMethods` dans les méthodes à l'aide des ressources qui ont été supprimées.
- Remplacez la déclaration `managedResource` par les déclarations des objets managés de votre classe qui doivent être supprimés. Si une classe implémente **IDisposable** ou contient une méthode **Close**, elle doit probablement être supprimée. Dans la méthode **Dispose**, fermez ou supprimez ces objets.
- Remplacez la déclaration `unManagedResource` par les déclarations des objets non managés de votre classe qui doivent être supprimés. La méthode de suppression de ces objets dépend de la définition de l'objet. Pour plus d'informations, consultez la documentation sur l'objet.

Programmation fiable

Une fois que la méthode **Dispose** a été appelée, les objets contenus dans votre collection ne sont pas valides. Vous devez tester le champ `disposed` avant d'exécuter des opérations sur votre objet. Pour obtenir un exemple, consultez la méthode `AnyOtherMethods` dans l'exemple de code.

Voir aussi

Référence

IDisposable

Concepts

Implémentation d'une méthode Dispose

Autres ressources

Garbage collection

Utilisation de constructeurs et de destructeurs

Les constructeurs et les destructeurs régissent la création et la destruction des objets.

Constructeurs

Pour créer un constructeur d'une classe, créez une procédure nommée **Sub New** à n'importe quel endroit de la définition de la classe. Pour créer un constructeur paramétré, spécifiez les noms et les types de données des arguments de la procédure **Sub New**, de la même façon que vous définiriez les arguments de toute autre procédure, comme dans le code ci-dessous :

VB

```
Sub New(ByVal s As String)
```

Les constructeurs sont souvent surchargés, comme l'illustre le code suivant :

VB

```
Sub New(ByVal s As String, i As Integer)
```

Quand vous définissez une classe dérivée d'une autre classe, la première ligne d'un constructeur doit constituer un appel au constructeur de la classe de base, à moins que la classe de base ne possède un constructeur accessible n'acceptant pas de paramètre. Un appel à la classe de base contenant le constructeur ci-dessus, par exemple, serait `MyBase.New(s)`. Sinon, `MyBase.New` est facultatif, et le runtime Visual Basic l'appelle implicitement.

À la suite du code qui appelle le constructeur de l'objet parent, vous pouvez éventuellement ajouter du code d'initialisation supplémentaire à la procédure **Sub New**. **Sub New** peut accepter des arguments lorsqu'il est appelé en tant que constructeur paramétré. Ces paramètres sont passés à partir de la procédure qui appelle le constructeur, comme dans `Dim AnObject As New ThisClass(X)`.

Destructeurs

Le code suivant montre comment utiliser **Dispose** et **Finalize** pour libérer des ressources dans une classe de base.

Remarque

Vous devez suivre les instructions pour implémenter IDisposable défini dans Durée de vie d'un objet : création et destruction des objets.

VB

```
    ' Design pattern for a base class.
    Public Class Base
        Implements IDisposable

        ' Keep track of when the object is disposed.
        Protected disposed As Boolean = False

        ' This method disposes the base object's resources.
        Protected Overridable Sub Dispose(ByVal disposing As Boolean)
            If Not Me.disposed Then
                If disposing Then
                    ' Insert code to free unmanaged resources.
                End If
                ' Insert code to free shared resources.
            End If
            Me.disposed = True
        End Sub

#Region " IDisposable Support "
        ' Do not change or add Overridable to these methods.
        ' Put cleanup code in Dispose(ByVal disposing As Boolean).
        Public Sub Dispose() Implements IDisposable.Dispose
            Dispose(True)
```

```
            GC.SuppressFinalize(Me)
        End Sub
        Protected Overrides Sub Finalize()
            Dispose(False)
            MyBase.Finalize()
        End Sub
#End Region
    End Class
```

Le code suivant montre comment utiliser **Dispose** et **Finalize** pour libérer des ressources dans une classe dérivée.

VB

```
' Design pattern for a derived class.
Public Class Derived
    Inherits Base

    ' This method disposes the derived object's resources.
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If Not Me.disposed Then
            If disposing Then
                ' Insert code to free unmanaged resources.
            End If
            ' Insert code to free shared resources.
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' The derived class does not have a Finalize method
    ' or a Dispose method with parameters because it inherits
    ' them from the base class.
End Class
```

Le code suivant présente un modèle de design commun pour le destructeur **Dispose**, à l'aide d'un bloc **Using** et d'un bloc **Try...Finally** équivalent.

VB

```
Sub DemonstrateUsing()
    Using d As New Derived
        ' Code to use the Derived object goes here.
    End Using
End Sub

Sub DemonstrateTry()
    Dim d As Derived = Nothing
    Try
        d = New Derived
        ' Code to use the Derived object goes here.
    Finally
        ' Call the Dispose method when done, even if there is an exception.
        If Not d Is Nothing Then
            d.Dispose()
        End If
    End Try
End Sub
```

L'exemple suivant crée un objet à l'aide d'un constructeur paramétré, puis appelle des destructeurs une fois que l'objet a perdu son utilité.

Remarque

Bien que cet exemple utilise Collect pour indiquer les méthodes appelées par le garbage collector pour supprimer des méthodes, il est préférable généralement de laisser le Common Language Runtime (CLR) gérer l'opération garbage collection.

VB

```
Sub TestConstructorsAndDestructors()
    ' Demonstrate how the Using statement calls the Dispose method.
    Using AnObject As New ThisClass(6)
        ' Place statements here that use the object.
        MsgBox("The value of ThisProperty after being initialized " & _
        " by the constructor is " & AnObject.ThisProperty & ".")
    End Using

    ' Demonstrate how the garbage collector calls the Finalize method.
    Dim AnObject2 As New ThisClass(6)
    AnObject2 = Nothing
    GC.Collect()
End Sub

Public Class BaseClass
    Sub New()
        MsgBox("BaseClass is initializing with Sub New.")
    End Sub

    Protected Overrides Sub Finalize()
        MsgBox("BaseClass is shutting down with Sub Finalize.")
        ' Place final cleanup tasks here.
        MyBase.Finalize()
    End Sub
End Class

Public Class ThisClass
    Inherits BaseClass
    Implements IDisposable

    Sub New(ByVal SomeValue As Integer)
        ' Call MyBase.New if this is a derived class.
        MyBase.New()
        MsgBox("ThisClass is initializing with Sub New.")
        ' Place initialization statements here.
        ThisPropertyValue = SomeValue
    End Sub

    Private ThisPropertyValue As Integer
    Property ThisProperty() As Integer
        Get
            CheckIfDisposed()
            ThisProperty = ThisPropertyValue
        End Get
        Set(ByVal Value As Integer)
            CheckIfDisposed()
            ThisPropertyValue = Value
        End Set
    End Property

    Protected Overrides Sub Finalize()
        MsgBox("ThisClass is shutting down with Sub Finalize.")
        Dispose(False)
    End Sub

    ' Do not add Overridable to this method.
    Public Overloads Sub Dispose() Implements IDisposable.Dispose
        MsgBox("ThisClass is shutting down with Sub Dispose.")
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

    Private disposed As Boolean = False
    Public Sub CheckIfDisposed()
        If Me.disposed Then
            Throw New ObjectDisposedException(Me.GetType().ToString, _
```

```
            "This object has been disposed.")
        End If
    End Sub

    Protected Overridable Overloads Sub Dispose( _
    ByVal disposing As Boolean)
        MsgBox("ThisClass is shutting down with the Sub Dispose overload.")
        ' Place final cleanup tasks here.
        If Not Me.disposed Then
            If disposing Then
                ' Dispose of any managed resources.
            End If
            ' Dispose of any unmanaged resource.

            ' Call MyBase.Finalize if this is a derived class,
            ' and the base class does not implement Dispose.
            MyBase.Finalize()
        End If
        Me.disposed = True
    End Sub

End Class
```

Lorsque vous exécutez cet exemple, la classe `ThisClass` appelle le constructeur **Sub New** de la classe `BaseClass`. Au terme de l'exécution du constructeur de la classe de base, la classe `ThisClass` exécute les instructions restantes de **Sub New** qui initialisent une valeur pour la propriété `ThisProperty`.

Quand la classe n'est plus nécessaire, le destructeur **Dispose** est appelé dans `ThisClass`.

Cet exemple affiche les éléments suivants :

```
BaseClass is initializing with Sub New.

ThisClass is initializing with Sub New.

The value of ThisProperty after being initialized by the constructor is 6.

ThisClass is shutting down with Sub Dispose.

ThisClass is shutting down with the Sub Dispose overload.

BaseClass is shutting down with Sub Finalize.

BaseClass is initializing with Sub New.

ThisClass is initializing with Sub New.

ThisClass is shutting down with Sub Finalize.

ThisClass is shutting down with the Sub Dispose overload.

BaseClass is shutting down with Sub Finalize.
```

Voir aussi

Référence

Dispose

Concepts

Durée de vie d'un objet : création et destruction des objets

Destructeurs et méthodes Finalize

Fonctionnement des méthodes New et Finalize dans une hiérarchie de classe

Propriétés, champs et méthodes de classe

En général, les champs et les propriétés représentent des informations au sujet d'un objet, tandis que les méthodes représentent des actions qu'un objet peut effectuer.

Les rubriques suivantes décrivent les mécanismes d'ajout de propriétés, de champs et de méthodes aux classes, et traitent de questions relatives à ces éléments.

Dans cette section

Comment : ajouter des champs et des propriétés à une classe

Fournit des informations sur la déclaration de champs et de propriétés.

Propriétés et procédures Property

Explique le fonctionnement des procédures **Property** et la manière d'implémenter des types de propriété courants.

Procédures Property et champs

Explique quand il faut utiliser des champs pour stocker des données dans une classe et quand les propriétés se révèlent une meilleure option.

Méthodes de classe

Analyse les procédures publiques que vous ajoutez à une classe.

Propriétés et méthodes

Vous aide à déterminer si la fonctionnalité que vous désirez est mieux implémentée sous la forme d'une propriété ou d'une méthode.

Propriétés par défaut

Traite de propriétés utilisées quand aucune propriété spécifique n'est nommée.

Propriétés et méthodes surchargées

Explique la définition de plusieurs propriétés ou méthodes de même nom, mais utilisant des types de données différents.

Substitution de propriétés et de méthodes

Explique comment redéfinir une méthode ou une propriété héritée.

Rubriques connexes

Création et utilisation d'objets

Illustre la création et l'utilisation d'instances de classes.

Événements en Visual Basic

Illustre la déclaration et l'utilisation des événements.

Délégués en Visual Basic

Illustre la déclaration et l'utilisation des délégués.

Gestion de vos objets avec les collections

Fournit des informations sur l'utilisation de collections pour stocker et récupérer des objets.

Comment : ajouter des champs et des propriétés à une classe

Vous pouvez avoir recours aux champs et aux propriétés pour stocker des informations dans un objet. Bien que les champs et les propriétés soient difficiles à distinguer du point de vue de l'application cliente, ils diffèrent eu égard à la manière dont ils sont déclarés au sein d'une classe. Les champs sont tout simplement des variables publiques exposées par une classe, alors que les propriétés utilisent des procédures **Property** pour contrôler la façon dont les valeurs sont définies ou retournées.

Pour ajouter un champ à une classe

- Déclarez une variable publique dans la définition de la classe, comme l'illustre le code suivant :

  VB

```
Class ThisClass
    Public ThisField As String
End Class
```

Pour ajouter une propriété à une classe

1. Déclarez une variable locale dans la classe pour y stocker la valeur de propriété. Cette étape est requise car les propriétés en elles-mêmes n'allouent pas de l'espace de stockage. Pour empêcher que leurs valeurs ne soient modifiées directement, les variables employées pour stocker des valeurs de propriété doivent être déclarées **Private**.
2. Faites précéder les déclarations de propriété par des modificateurs, comme **Public** et **Shared**, en fonction des cas. Utilisez le mot clé **Property** pour déclarer le nom de la propriété et déclarez le type de données que la propriété stocke et retourne.
3. Définissez des procédures de propriétés **Get** et **Set** dans la définition de propriété. Les procédures de propriétés **Get** s'emploient pour retourner la valeur d'une propriété et sont à peu près l'équivalent des fonctions du point de vue de la syntaxe. Elles n'acceptent pas d'arguments et peuvent être utilisées pour retourner la valeur de variables locales privées déclarées dans la classe utilisée pour stocker la valeur de propriété. Les procédures de propriétés **Set** sont utilisées pour définir la valeur d'une propriété ; elles ont un paramètre, généralement nommé `Value`, avec le même type de données que la propriété elle-même. Chaque fois que la valeur de la propriété change, le paramètre `Value` est passé à la procédure de propriété **Set**, où il peut être validé et stocké dans une variable locale.
4. Terminez les procédures de propriétés **Get** et **Set** avec des instructions **End Get** et **End Set** selon les besoins.
5. Terminez le bloc de propriété par une instruction **End Property**.

> **Remarque**
>
> Si vous travaillez dans l'environnement de développement intégré (IDE) Visual Studio, vous pouvez faire en sorte qu'il crée des procédures de propriétés **Get** et **Set** vides. Tapez **Property** *PropName* **As** *DataType* (où *PropName* est le nom de la propriété et *DataType* est un type de données spécifique, tel que **Integer**), et les procédures de propriétés apparaissent dans l'**Éditeur de code**.

L'exemple ci-dessous déclare une propriété dans une classe :

VB

```
Class ThisClass
    Private m_PropVal As String
    Public Property One() As String
        Get
            ' Return the value stored in the local variable.
            Return m_PropVal
        End Get
        Set(ByVal Value As String)
            ' Store the value in a local variable.
            m_PropVal = Value
```

```
        End Set
    End Property
End Class
```

Lorsque vous créez une instance de `ThisClass` et définissez la valeur de la propriété `One`, la procédure de propriété **Set** est appelée et la valeur est passée dans le paramètre `Value`, qui est stocké dans une variable locale appelée `m_PropVal`. Quand la valeur de cette propriété est extraite, la procédure de propriété **Get** est appelée à la manière d'une fonction et retourne la valeur stockée dans la variable locale `m_PropVal`.

Voir aussi

Tâches

Comment : ajouter des événements à une classe

Référence

Property, instruction

Public (Visual Basic)

Shared (Visual Basic)

Concepts

Propriétés et procédures Property

Propriétés et procédures Property

Vous pouvez avoir recours aux propriétés et aux champs pour stocker des informations dans un objet. Les champs sont tout simplement des variables publiques, alors que les propriétés utilisent des procédures de propriété pour contrôler la façon dont les valeurs sont définies ou retournées. Les procédures de propriété sont des blocs de code déclarés dans des définitions de propriété que vous pouvez utiliser pour exécuter du code quand une valeur de propriété est définie ou extraite.

Visual Basic possède deux types de procédures de propriété : les procédures de propriété **Get** qui permettent d'extraire une valeur de propriété, et les procédures de propriété **Set** qui servent à assigner une valeur à une propriété. Par exemple, une propriété qui stocke le solde d'un compte bancaire peut utiliser le code d'une procédure de propriété **Get** pour imputer les intérêts et contrôler les frais de gestion avant de retourner le solde disponible. Vous pouvez utiliser par la suite la procédure de propriété **Set** pour valider le solde et éviter qu'elle soit mise à jour de façon incorrecte. En bref, les procédures de propriété permettent à un objet de protéger et de valider ses propres données.

Le fragment de code suivant compare la manière dont les champs et les propriétés valident les valeurs.

VB

```
Protected Sub TestFieldsAndProperties()
    ' Assume, for this example, that the only valid values for
    ' the field and property are numbers less than 10.
    Dim NewClass As New ThisClass

    ' Test data validation.

    ' Works because there is no data validation.
    NewClass.ThisField = 36
    ' Will print 36.
    MsgBox("ThisField = " & NewClass.ThisField)

    ' The attempt to set the field to a value greater than 10 will silently fail.
    NewClass.ThisProperty = 36
    ' The next statement will print the old value of 0 instead.
    MsgBox("ThisProperty = " & NewClass.ThisProperty)
End Sub

Public Class ThisClass
    ' Declare a field.
    Public ThisField As Integer
    ' Field used for Property Set operations.
    Private thisPropertyValue As Integer = 0
    ' Declare a property.
    Public Property ThisProperty() As Integer
        Get
            Return thisPropertyValue
        End Get
        Set(ByVal Value As Integer)
            ' Only allow Set operation for values less than 10.
            If Value < 10 Then thisPropertyValue = Value
        End Set
    End Property
End Class
```

La procédure `TestFieldsAndProperties` crée une instance de la classe, puis définit et extrait les valeurs des champs et des propriétés. Pour les besoins de l'exemple, nous supposons que seuls les nombres inférieurs à 10 sont des valeurs valides. Étant donné qu'aucune méthode ne permet de valider les valeurs assignées aux champs, l'attribution de la valeur 36 au champ est admise. Cependant, la propriété effectue des assignations uniquement pour les nombres inférieurs à 10, c'est pourquoi la tentative de lui attribuer la valeur 36 est ignorée.

Propriétés en lecture seule et en écriture seule

La plupart des propriétés possèdent à la fois des procédures de propriété **Get** et **Set** pour vous permettre de lire et de modifier la valeur qu'elles stockent. Vous pouvez néanmoins utiliser les modificateurs **ReadOnly** ou **WriteOnly** pour empêcher la modification ou la lecture des propriétés.

Les propriétés en lecture seule ne peuvent pas être assorties de procédures de propriété **Set** ; ces propriétés sont utiles pour les éléments que vous voulez exposer sans qu'ils ne puissent être modifiés. Par exemple, vous pouvez recourir à une propriété en lecture seule qui fournira la vitesse de processeur d'un ordinateur.

Les propriétés en écriture seule ne peuvent pas avoir de procédures de propriété **Get** et sont utiles pour configurer des objets dont les données ne doivent pas ou ne peuvent pas être stockées dans l'objet. Par exemple, une propriété en écriture seule peut être utilisée pour gérer un mot de passe et modifier l'état de l'objet sans stocker le mot de passe.

Remarque
Les versions précédentes de Visual Basic prennent en charge des procédures de propriété **Let** qui peuvent être utilisées lors de l'assignation des objets aux propriétés. Dans Visual Basic, les procédures de propriété **Let** ne sont plus nécessaires puisque que l'assignation d'objets est gérée comme tout autre type d'assignation.

Voir aussi

Référence

Get, instruction
Set, instruction (Visual Basic)
ReadOnly (Visual Basic)
WriteOnly

Concepts

Procédures Property
Procédures Property et champs

Procédures Property et champs

Les propriétés et les champs stockent et extraient des informations dans un objet. En raison de leur similitude, il peut être difficile pour le programmeur de déterminer laquelle des deux convient le mieux dans une situation donnée.

Utilisez des procédures Property si :

- vous devez contrôler quand et comment une valeur est définie ou extraite ;
- la propriété possède un ensemble bien défini de valeurs devant être validées ;
- le fait de définir la valeur entraîne une modification perceptible de l'état de l'objet, comme pour une propriété `IsVisible` ;
- le fait de définir la propriété entraîne la modification d'autres variables internes ou d'autres valeurs d'autres propriétés ;
- une série d'étapes doivent être réalisées avant que la propriété ne puisse être définie ou extraite.

Utilisez des champs si :

- la valeur est d'un type à validation automatique, comme par exemple lorsqu'une erreur ou la conversion automatique de données a lieu si une valeur différente de **True** ou **False** est assignée à une variable **Boolean** ;
- toute valeur située dans la plage de valeurs prises en charge par le type de données est valide, ce qui est le cas pour de nombreuses propriétés du type **Single** ou **Double** ;
- la propriété est du type de données **String** et s'il n'existe aucune contrainte quant à la taille ou à la valeur de la chaîne.

Voir aussi

Concepts

Classes et modules
Propriétés et procédures Property
Types de données en Visual Basic

Méthodes de classe

Les méthodes d'une classe sont les procédures **Sub** ou **Function** déclarées dans la classe. Par exemple, pour créer une méthode `Withdrawal` d'une classe intitulée `Account`, vous pouvez ajouter cette fonction **Public** au module de classe :

VB

```
Public Function WithDrawal(ByVal Amount As Decimal, _
      ByVal TransactionCode As Byte) As Double
    ' Add code here to perform the withdrawal,
    ' return a transaction code,
    ' or to raise an overdraft error.
End Function
```

Méthodes partagées

Les méthodes partagées peuvent être directement appelées à partir d'une classe sans devoir créer préalablement une instance de la classe. Les méthodes partagées sont utiles lorsque vous ne voulez pas associer une méthode à une instance spécifique d'une classe. Les méthodes partagées ne peuvent pas être déclarées à l'aide des modificateurs **Overridable**, **NotOverridable** et **MustOverride**. Les méthodes déclarées dans des modules sont implicitement partagées et ne peuvent pas explicitement utiliser le modificateur **Shared**.

Exemple

VB

```
Class ShareClass
    Shared Sub SharedSub()
        MsgBox("Shared method.")
    End Sub
End Class

Sub Test()
    ' Call the method.
    ShareClass.SharedSub()
End Sub
```

Protection des détails d'implémentation

Les procédures d'utilitaire employées en interne par une classe doivent être déclarées comme **Private**, **Protected**, **Friend**. La limitation de l'accès aux méthodes de ce type protège les développeurs qui utilisent vos objets en vous permettant d'y apporter des modifications ultérieures sans affecter le code qui utilise les objets.

La protection des détails d'implémentation d'un objet est un autre aspect de l'*encapsulation*. L'encapsulation permet d'améliorer les performances des méthodes ou de modifier totalement l'implémentation d'une méthode, sans avoir besoin de modifier le code qui utilise la méthode.

Voir aussi

Tâches

Comment : ajouter des événements à une classe

Référence

Overridable
NotOverridable
MustOverride
Shared (Visual Basic)
Public (Visual Basic)
Private (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)

Concepts

Propriétés et méthodes
Membres partagés en Visual Basic

Propriétés et méthodes

Les propriétés et les méthodes sont similaires dans la mesure où elles sont implémentées en tant que procédures qui acceptent des arguments. En règle générale, les propriétés stockent des données pour un objet, tandis que les méthodes sont des actions susceptibles d'être effectuées par un objet. Certaines caractéristiques d'un objet sont des propriétés (par exemple, `Name`), d'autres sont des méthodes (par exemple, `Move` et `Show`). Il n'est pas toujours facile de déterminer les membres de classe qui doivent être des propriétés et ceux qui doivent être des méthodes. Par exemple, la méthode `Item` d'une classe de collection stocke et extrait des données et peut être implémentée en tant que propriété indexée. Il est toutefois recommandé d'implémenter `Item` en tant que méthode.

Syntaxe de propriété et syntaxe de méthode

Pour déterminer la façon d'implémenter un membre de classe, vous devez tenir compte de son utilisation. Même si la syntaxe permettant d'extraire des informations d'une propriété paramétrée est pratiquement identique à la syntaxe utilisée avec une méthode implémentée en tant que fonction, la syntaxe permettant de modifier une telle valeur est légèrement différente. Par exemple, si vous implémentez le membre d'une classe en tant que propriété, la syntaxe suivante décrit la manière dont il doit être utilisé :

ThisObject.*ThisProperty*(*Index*) = *NewValue*

Si vous implémentez le membre de classe en tant que méthode, la valeur modifiée doit être un argument. Le fragment de code suivant décrit une syntaxe équivalente :

ThisObject.*ThisProperty*(*Index*, *NewValue*)

Messages d'erreur

Lors de l'implémentation d'un membre de classe, vous devez également tenir compte du type de message généré en cas d'utilisation incorrecte de la classe. Si quelqu'un tente accidentellement d'assigner une valeur à une propriété en lecture seule, un message d'erreur différent est retourné en réponse à un appel identique de méthode. Les membres de classe correctement implémentés retournent des messages d'erreur plus simples à interpréter.

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Référence

Item, propriété (objet Collection)

Concepts

Méthodes de classe

Propriétés par défaut

Une propriété qui accepte des arguments peut être déclarée comme la propriété par défaut d'une classe. Une *propriété par défaut* est la propriété utilisée par Visual Basic si aucune propriété spécifique n'a été désignée pour un objet. Les propriétés par défaut sont utiles, car elles vous permettent de condenser davantage votre code source en omettant les noms de propriétés fréquemment utilisés.

Les propriétés qui se prêtent le mieux à devenir des propriétés par défaut sont celles qui acceptent des paramètres et qui sont susceptibles d'être utilisées fréquemment. Par exemple, la propriété `Item` représente une propriété par défaut judicieuse pour une classe de collection étant donné qu'elle est fréquemment utilisée.

Les règles suivantes s'appliquent aux propriétés par défaut :

- Un type ne peut compter qu'une seule propriété par défaut, y compris des propriétés héritées d'une classe de base. Il existe une exception à cette règle. Une propriété par défaut définie dans une classe de base peut être occultée par une autre propriété par défaut dans une classe dérivée.
- Si une propriété par défaut dans une classe de base est occultée par une propriété non par défaut dans une classe dérivée, la propriété par défaut reste accessible à l'aide de la syntaxe de propriété par défaut.
- Une propriété par défaut ne peut pas être **Shared** ni **Private**.
- Si une propriété surchargée est une propriété par défaut, toutes les propriétés surchargées du même nom doivent également spécifier **Default**.
- Les propriétés par défaut doivent accepter au moins un argument.

Exemple

L'exemple suivant déclare une propriété qui contient un tableau de chaînes en tant que propriété par défaut d'une classe :

VB

```
Class Class2
    ' Define a local variable to store the property value.
    Private PropertyValues As String()
    ' Define the default property.
    Default Public Property Prop1(ByVal Index As Integer) As String
        Get
            Return PropertyValues(Index)
        End Get
        Set(ByVal Value As String)
            If PropertyValues Is Nothing Then
                ' The array contains Nothing when first accessed.
                ReDim PropertyValues(0)
            Else
                ' Re-dimension the array to hold the new element.
                ReDim Preserve PropertyValues(UBound(PropertyValues) + 1)
            End If
            PropertyValues(Index) = Value
        End Set
    End Property
End Class
```

Accès aux propriétés par défaut

Vous pouvez accéder aux propriétés par défaut en employant une syntaxe abrégée. Par exemple, le fragment de code suivant utilise la syntaxe standard et la syntaxe de la propriété par défaut :

VB

```
Dim C As New Class2
' The first two lines of code access a property the standard way.

' Property assignment.
```

```
C.Prop1(0) = "Value One"
' Property retrieval.
MsgBox(C.Prop1(0))

' The following two lines of code use default property syntax.

' Property assignment.
C(1) = "Value Two"
' Property retrieval.
MsgBox(C(1))
```

Voir aussi

Référence

Default (Visual Basic)

Concepts

Recommandations de mise à niveau : résoudre les propriétés par défaut sans paramètres

Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0

Propriétés et méthodes surchargées

La surcharge consiste à créer plusieurs procédures, constructeurs d'instance ou propriétés dans une classe avec le même nom, mais avec des types d'argument différents.

Utilisation de la surcharge

La surcharge est particulièrement utile lorsque votre modèle objet impose l'utilisation de noms identiques pour des procédures qui s'appliquent à des types de données différents. Par exemple, les procédures `Display` d'une classe pouvant afficher plusieurs types de données distincts peuvent se présenter comme suit :

VB

```
Overloads Sub Display(ByVal theChar As Char)
    ' Add code that displays Char data.
End Sub
Overloads Sub Display(ByVal theInteger As Integer)
    ' Add code that displays Integer data.
End Sub
Overloads Sub Display(ByVal theDouble As Double)
    ' Add code that displays Double data.
End Sub
```

Sans surcharge, vous devez créer des noms distincts pour chacune des procédures, même si celles-ci effectuent les mêmes tâches, comme le montre l'exemple suivant :

VB

```
Sub DisplayChar(ByVal theChar As Char)
    ' Add code that displays Char data.
End Sub
Sub DisplayInt(ByVal theInteger As Integer)
    ' Add code that displays Integer data.
End Sub
Sub DisplayDouble(ByVal theDouble As Double)
    ' Add code that displays Double data.
End Sub
```

La surcharge facilite l'utilisation des propriétés ou des méthodes, car elle propose un choix de divers types de données disponibles. Par exemple, la méthode `Display` surchargée décrite ci-dessus peut être appelée par l'une des lignes de code suivantes :

VB

```
' Call Display with a literal of type Char.
Display("9"c)
' Call Display with a literal of type Integer.
Display(9)
' Call Display with a literal of type Double.
Display(9.9R)
```

Au moment de l'exécution, Visual Basic appelle la procédure correcte en fonction des types de données des paramètres spécifiés.

Remarque

La surcharge, la substitution et l'occultation sont des concepts similaires pouvant être facilement confondus. Pour plus d'informations, consultez Introduction aux objets dans Visual Basic.

Règles de surcharge

Pour créer un membre surchargé d'une classe, ajoutez au moins deux propriétés ou méthodes portant le même nom. Hormis les membres dérivés surchargés, chaque membre surchargé doit posséder des listes distinctes de paramètres, et les éléments suivants ne peuvent pas être utilisés pour opérer une distinction lors de la surcharge d'une propriété ou d'une procédure :

- modificateurs (par exemple **ByVal** ou **ByRef**) qui s'appliquent à un membre ou à des paramètres du membre ;
- noms de paramètres ;
- types de retour de procédures.

Le mot clé **Overloads** est facultatif lors de la surcharge, mais si un membre surchargé l'utilise, tous les autres membres surchargés portant le même nom doivent également spécifier ce mot clé.

Les classes dérivées peuvent surcharger les membres hérités avec des membres dont les paramètres et les types de paramètre sont identiques ; ce processus porte le nom d'*occultation par le nom et la signature*. Si le mot clé **Overloads** est utilisé lors de l'occultation par le nom et la signature, l'implémentation de la classe dérivée du membre sera utilisée à la place de l'implémentation dans la classe de base, et toutes les autres surcharges de ce membre seront disponibles pour les instances de la classe dérivée.

Si le mot clé **Overloads** est omis lors de la surcharge d'un membre hérité avec un membre dont les paramètres et les types de paramètre sont identiques, la surcharge est appelée l'*occultation par le nom*. L'occultation par le nom remplace l'implémentation héritée d'un membre et rend toutes les autres surcharges indisponibles pour les instances de la classe dérivée et ses descendants.

Les modificateurs **Overloads** et **Shadows** ne peuvent pas être utilisés avec la même propriété ou méthode.

Exemple

L'exemple suivant crée des méthodes surchargées qui acceptent une valeur de type **String** ou **Decimal** correspondant à un montant en dollars et qui retournent une chaîne comportant le taux de T.V.A.

Pour utiliser cet exemple afin de créer une méthode surchargée

1. Ouvrez un nouveau projet et ajoutez une classe appelée `TaxClass`
2. Ajoutez le code suivant à la classe `TaxClass` :

   VB

```
Public Class TaxClass
    Overloads Function TaxAmount(ByVal decPrice As Decimal, _
         ByVal TaxRate As Single) As String
        TaxAmount = "Price is a Decimal. Tax is $" & _
           (CStr(decPrice * TaxRate))
    End Function

    Overloads Function TaxAmount(ByVal strPrice As String, _
          ByVal TaxRate As Single) As String
        TaxAmount = "Price is a String. Tax is $" & _
           CStr((CDec(strPrice) * TaxRate))
    End Function
End Class
```

3. Ajoutez la procédure suivante à votre formulaire :

   VB

```
Sub ShowTax()
    ' 8% tax rate.
    Const TaxRate As Single = 0.08
    ' $64.00 Purchase as a String.
    Dim strPrice As String = "64.00"
    ' $64.00 Purchase as a Decimal.
    Dim decPrice As Decimal = 64
    Dim aclass As New TaxClass
```

```
    'Call the same method with two different kinds of data.
    MsgBox(aclass.TaxAmount(strPrice, TaxRate))
    MsgBox(aclass.TaxAmount(decPrice, TaxRate))
End Sub
```

4. Ajoutez un bouton à votre formulaire et appelez la procédure `ShowTax` à partir de l'événement de bouton `Button1_Click`.

5. Exécutez le projet et cliquez sur le bouton du formulaire afin de tester la procédure surchargée `ShowTax`.

Au moment de l'exécution, le compilateur sélectionne la fonction surchargée adéquate qui correspond aux paramètres utilisés. Lorsque vous cliquez sur le bouton, la méthode surchargée est d'abord appelée avec un paramètre `Price` (une chaîne) ; le message "Price is a String. Tax is $5.12" s'affiche. `TaxAmount` est appelé avec une valeur **Decimal** la seconde fois et le message "Price is a Decimal. Tax is $5,12" s'affiche.

Voir aussi

Référence

Sub, instruction (Visual Basic)
Shadows
ByVal
ByRef
Overloads
Shadows

Concepts

Occultation dans Visual Basic
Introduction aux objets dans Visual Basic
Substitution de propriétés et de méthodes

Substitution de propriétés et de méthodes

Une classe dérivée hérite des propriétés et des méthodes définies dans sa classe de base. Cela est utile parce que vous pouvez réutiliser ces éléments s'ils sont appropriés pour la classe dérivée. Si la propriété ou la méthode dans la classe de base est marquée avec le mot clé **Overridable**, vous pouvez définir une nouvelle implémentation pour le membre dans la classe dérivée. Utilisez le mot clé **Overrides** pour occulter le membre en le redéfinissant dans la classe dérivée. Cette opération s'avère utile lorsque vous ne pouvez pas utiliser le membre « tel quel ».

En pratique, les membres substitués sont souvent utilisés pour implémenter le polymorphisme. Pour plus d'informations, consultez Polymorphisme.

Les règles suivantes s'appliquent à la substitution de méthodes :

- Vous ne pouvez substituer que les membres marqués du mot clé **Overridable** dans leur classe de base.
- Par défaut, les propriétés et les méthodes sont **NotOverridable**.
- Les membres substitués doivent avoir les mêmes arguments que les membres hérités de la classe de base.
- La nouvelle implémentation d'un membre peut appeler l'implémentation d'origine dans la classe parente en spécifiant **MyBase** avant le nom de la méthode.

> **Remarque**
>
> La surcharge, la substitution et l'occultation sont des concepts similaires pouvant être facilement confondus. Pour plus d'informations, consultez Introduction aux objets dans Visual Basic.

Exemple

Supposons que vous souhaitiez définir des classes pour gérer les salaires. Vous pouvez définir une classe générique `Payroll` qui contient une méthode `RunPayroll` permettant de calculer les salaires pour une semaine normale. Vous pouvez ensuite utiliser `Payroll` comme une classe de base pour une classe `BonusPayroll` plus spécialisée qui peut être utilisée lors de la distribution de primes aux employés.

La classe `BonusPayroll` peut hériter et substituer la méthode `PayEmployee` définie dans la classe de base `Payroll`.

L'exemple suivant définit une classe de base `Payroll`, et une classe dérivée `BonusPayroll` pour substituer une méthode héritée nommée `PayEmployee`. Une procédure, `RunPayroll`, crée et passe ensuite un objet `Payroll` et un objet `BonusPayroll` à une fonction, `Pay`, qui exécute la méthode `PayEmployee` sur les deux objets.

VB

```
Const BonusRate As Decimal = 1.45D
Const PayRate As Decimal = 14.75D

Class Payroll
    Overridable Function PayEmployee( _
        ByVal HoursWorked As Decimal, _
        ByVal PayRate As Decimal) _
        As Decimal

        PayEmployee = HoursWorked * PayRate
    End Function
End Class

Class BonusPayroll
    Inherits Payroll
    Overrides Function PayEmployee( _
        ByVal HoursWorked As Decimal, _
        ByVal PayRate As Decimal) _
        As Decimal

        ' The following code calls the original method in the base
        ' class, and then modifies the returned value.
        PayEmployee = MyBase.PayEmployee(HoursWorked, PayRate) * BonusRate
    End Function
```

```
End Class

Sub RunPayroll()
    Dim PayrollItem As Payroll = New Payroll
    Dim BonusPayrollItem As New BonusPayroll
    Dim HoursWorked As Decimal = 40

    MsgBox("Normal pay is: " & _
        PayrollItem.PayEmployee(HoursWorked, PayRate))
    MsgBox("Pay with bonus is: " & _
        BonusPayrollItem.PayEmployee(HoursWorked, PayRate))
End Sub
```

Voir aussi

Concepts

Propriétés et méthodes surchargées

Modificateurs Override

Occultation dans Visual Basic

Autres ressources

Polymorphisme

Modificateurs Override

Vous pouvez utiliser les modificateurs **NotOverridable** et **MustOverride** dans une classe de base pour contrôler la manière dont les classes dérivées substituent leurs propriétés et leurs méthodes.

Le modificateur **NotOverridable** définit une méthode d'une classe de base qui ne peut pas être substituée dans des classes dérivées. Toutes les méthodes sont **NotOverridable** sauf si elles sont marquées du modificateur **Overridable**. Vous pouvez faire appel au modificateur **NotOverridable** si vous ne voulez pas qu'une méthode substituée puisse à nouveau être substituée dans une classe dérivée.

Les méthodes définies par le modificateur **MustOverride** n'ont pas d'implémentation dans la classe de base et elles doivent être implémentées dans les classes dérivées. Les classes qui contiennent les méthodes **MustOverride** doivent être marquées par le modificateur **MustInherit**.

Exemple

VB

```
MustInherit Class BaseClass
    Public MustOverride Sub aProcedure()
End Class

Class DerivedClass
    Inherits BaseClass
    Public NotOverridable Overrides Sub aProcedure()
        ' Override a procedure inherited from the base class
        ' and mark it with the NotOverridable modifier so that
        ' it cannot be overridden in classes derived from this class.
    End Sub
End Class
```

Voir aussi

Concepts

Propriétés et méthodes surchargées

Création et utilisation d'objets

Les objets sont des combinaisons de code et de données qui peuvent être traitées comme des entités uniques. Quasiment toutes vos actions en Visual Basic reposent sur des objets, qui vont des contrôles que vous utilisez aux formulaires dans lesquels vous les insérez. Cette section traite de l'origine des objets et explique comment les utiliser.

Dans cette section

Objets issus de Visual Basic et d'autres sources

Explique comment utiliser des objets à partir de Microsoft Word, Microsoft Excel et d'autres applications.

Comment : définir et récupérer des propriétés

Explique comment modifier et obtenir des valeurs de propriétés.

Comment : exécuter des actions avec les méthodes

Explique ce que sont les méthodes et comment les utiliser.

Comment : traiter les formulaires comme des objets

Montre comment utiliser les formulaires comme des objets ordinaires.

Comment : accéder à un formulaire

Montrez comment accéder aux membres d'un formulaire sans avoir à créer de variable.

Comment : utiliser le nouveau mot clé

Montre comment créer des instances de formulaires, des classes définies dans des modules de classe et des collections.

Gestion des ressources

Explique comment diminuer les besoins en mémoire et en ressources système grâce à la libération des références aux objets.

Comment : passer des objets aux procédures

Explique comment passer des objets sous forme d'arguments aux procédures.

Procédure pas à pas : persistance d'un objet en Visual Basic

Montre comment utiliser la sérialisation pour rendre les données d'un objet persistantes entre les instances, ce qui vous permet de stocker des valeurs et de les récupérer lors de la prochaine instanciation de l'objet.

Comment : hériter d'une classe dans Visual Basic

Montre comment définir une classe de base et déclarer des classes qui étendent la classe de base.

Rubriques connexes

Objets en Visual Basic

Présente les notions de base relatives à la programmation orientée objet.

Gestion de groupes d'objets

Répertorie les techniques de travail sur les tableaux et les collections d'objets.

Comment : exécuter plusieurs actions sur un objet

Traite de l'utilisation de l'instruction **With...End With** pour simplifier l'accès à une propriété.

Objets issus de Visual Basic et d'autres sources

Les objets utilisés dans Visual Basic sont issus de sources internes et externes. Des exemples d'objets internes comprennent des objets intrinsèques et des classes de votre projet ; des exemples d'objets externes comprennent des assemblys et des objets COM.

Objets internes

Les objets *intrinsèques* (ou *intégrés*) sont des objets fournis par Visual Basic. Ils comprennent les types scalaires primitifs, tels que **Integer** et **Double** ainsi que les types **Array** et **String**. Vous ne devez pas créer de références à des objets internes avant de les utiliser dans votre projet.

D'autres objets internes sont des instances de classes dans votre projet en cours. Vous pouvez utiliser ces classes dès que vous en avez besoin dans votre projet et les rendre disponibles à d'autres applications lorsque vous créez un assembly.

Objets externes

Les objets externes sont issus d'autres projets ou assemblys qui ne sont pas disponibles par défaut dans votre projet. Vous devez créer des références de projet à des objets externes avant de les utiliser dans votre projet.

Les assemblys sont la source la plus ordinaire des objets pour les applications Visual Basic. Le .NET Framework comprend des assemblys qui contiennent les objets les plus fréquemment utilisés. Certains objets du .NET Framework sont traités comme des objets intrinsèques, mais la plupart des assemblys doivent être importés explicitement via l'instruction **Imports** avant de pouvoir être utilisés. Tout langage conforme CLS (Common Language Specification), tel que Visual Basic ou Visual C#, peut créer et utiliser des assemblys. Pour plus d'informations, consultez Assemblys.

Auparavant, les composants COM constituaient une source traditionnelle d'objets pour les programmeurs Visual Basic, mais à l'heure actuelle, les assemblys .NET Framework sont mieux adaptés aux nouveaux objets. Vous pouvez toujours utiliser des composants COM existants dans vos applications, mais vous devez accéder aux objets COM via les classes d'interopérabilité .NET Framework. L'accès à une bibliothèque COM exige l'utilisation d'un assembly d'interopérabilité contenant des classes d'interopérabilité pour chacune des classes COM définies dans la bibliothèque COM. Pour plus d'informations, consultez COM Interop.

En plus d'accéder aux classes .NET Framework natives et aux classes COM, il est également possible d'appeler des fonctions définies dans les bibliothèques de liens dynamiques (DLL), telles que celles de l'API Win32. Visual Basic vous permet d'appeler des fonctions dans les DLL après qu'elles aient été déclarés à l'aide d'une instruction **Declare**. Visual Basic ajoute des fonctionnalités à l'instruction **Declare** en vous permettant d'utiliser l'attribut DllImportAttribute pour spécifier des valeurs par défaut pour les éléments, tels que CallingConvention, ExactSpelling et SetLastError. Les paramètres des instructions **Declare** peuvent contenir l'attribut MarshalAsAttribute qui prend en charge la conversion de paramètres de façon différente par rapport aux versions antérieures de Visual Basic.

Voir aussi

Tâches

Procédure pas à pas : implémentation de l'héritage avec les objets COM
Comment : définir et récupérer des propriétés

Référence

Declare, instruction
Imports, instruction
Integer, type de données (Visual Basic)
Double, type de données (Visual Basic)
String, type de données (Visual Basic)
DllImportAttribute
MarshalAsAttribute

Concepts

Relations entre les objets

Autres ressources

Création et utilisation d'objets

Comment : définir et récupérer des propriétés

Lorsque vous utilisez des formulaires et des contrôles dans Visual Basic, vous pouvez définir leurs propriétés par programme au moment de l'exécution ou en mode design à l'aide de la fenêtre **Propriétés**. Les propriétés de la plupart des autres objets, issus par exemple d'un assembly ou d'un objet, ne peuvent être définies que par programme.

Les propriétés que vous pouvez définir et lire sont dites *en lecture-écriture*. Les propriétés que vous pouvez lire, mais pas modifier sont dites *en lecture seule*. Les propriétés que vous pouvez écrire, mais pas lire sont dites *en écriture seule*.

Vous définissez la valeur d'une propriété lorsque vous voulez modifier le comportement ou l'aspect d'un objet. Par exemple, vous modifiez la propriété `Text` d'un contrôle de zone de texte pour modifier le contenu de cette zone.

Vous extrayez la valeur d'une propriété lorsque vous voulez connaître l'état d'un objet avant que le code n'accomplisse d'autres actions telles qu'assigner la valeur à un autre objet. Par exemple, vous pouvez retourner la propriété `Text` d'un contrôle de zone de texte pour déterminer le contenu de celle-ci avant d'exécuter un code susceptible de modifier sa valeur.

Pour définir la valeur des propriétés

- Utilisez la syntaxe suivante :

  Object **.** *property* **=** *expression*

  Les exemples d'instructions suivants montrent comment définir les propriétés :

  VB

```
' Set the Top property to 200 twips.
TextBox1.Top = 200
' Display the text box.
TextBox1.Visible = True
' Display 'hello' in the text box.
TextBox1.Text = "hello"
```

> **Remarque**
>
> Vous pouvez également définir une propriété en la passant à des paramètres **ByRef**, auquel cas celle-ci est modifiée pa r le résultat retourné par le paramètre **ByRef**.

Pour extraire les valeurs de propriétés

- Utilisez la syntaxe suivante :

  variable **=** *Object***.***property*

  Vous pouvez également extraire la valeur d'une propriété dans une expression plus complexe sans assigner la propriété à une variable. Le code suivant modifie la propriété **Top** d'un contrôle de case d'option :

  VB

```
RadioButton1.Top += 20
```

Voir aussi

Tâches

Comment : exécuter des actions avec les méthodes

Concepts

Objets issus de Visual Basic et d'autres sources

Autres ressources

Création et utilisation d'objets

Comment : exécuter des actions avec les méthodes

Les méthodes sont des procédures associées à des objets. À la différence des champs et des propriétés, qui représentent les informations que peut enregistrer un objet, les méthodes représentent les actions qu'un objet peut accomplir. Les méthodes peuvent affecter la valeur des propriétés. Par exemple, pour faire une analogie avec un poste de radio, vous pouvez utiliser une méthode `SetVolume` pour modifier la valeur d'une propriété `Volume`. De la même façon, dans Visual Basic, les éléments de zones de liste ont une propriété `List` que vous pouvez modifier avec les méthodes `Clear` et `Add`.

Lorsque vous utilisez un argument dans le code, vous écrivez l'instruction en tenant compte du nombre d'arguments requis par la méthode et du fait qu'elle retourne ou non une valeur. En général, vous utilisez les méthodes comme vous utilisez des sous-routines ou des appels de fonction. Mais vous pouvez aussi appeler des méthodes comme vous appelez des procédures de module, à cette exception près que vous pouvez qualifier les méthodes avec une expression spécifiant l'instance d'objet dont la méthode doit être appelée. En l'absence de qualification, la variable **Me** est implicitement utilisée comme instance.

Pour utiliser une méthode ne requérant pas d'arguments

- Utilisez la syntaxe suivante :

  Object.method()

  Dans l'exemple suivant, la méthode `Refresh` repeint la zone d'image :

  VB

```
' Force the control to repaint.
PictureBox1.Refresh()
```

> **Remarque**
>
> Certaines méthodes, par exemple `Refresh`, n'ont pas d'arguments et ne retournent pas de valeurs.

Pour utiliser une méthode requérant plusieurs arguments

- Placez les arguments entre parenthèses en les séparant par des virgules. Dans l'exemple suivant, la méthode **MsgBox** utilise des arguments spécifiant le message à afficher et le style du message :

  VB

```
MsgBox("Database update complete", _
       MsgBoxStyle.OKOnly Or MsgBoxStyle.Exclamation, _
       "My Application")
```

Pour utiliser une méthode retournant une valeur

- Assignez la valeur de retour à une variable ou utilisez directement l'appel de méthode comme un paramètre d'un autre appel. Le code suivant stocke la valeur de retour :

  VB

```
Dim Response As MsgBoxResult
Response = MsgBox("Do you want to exit?", _
                  MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
                  "My Application")
```

  Cet exemple utilise la valeur retournée par la méthode **Len** comme argument de **MsgBox**.

  VB

```
Dim TestStr As String = "Some String"
' Display the string "String length is : 11".
MsgBox("String length is : " & Len(TestStr))
```

Voir aussi

Tâches

Comment : définir et récupérer des propriétés

Concepts

Relations entre les objets

Autres ressources

Création et utilisation d'objets

Comment : traiter les formulaires comme des objets

Les formulaires sont les objets graphiques qui constituent l'interface utilisateur de vos applications. Dans Visual Basic, les classes définissent le mode d'affichage des formulaires et les actions que ceux-ci peuvent effectuer. Lorsqu'un formulaire est affiché au moment de l'exécution, Visual Basic crée une instance de la classe Form que vous pouvez utiliser comme n'importe quel autre objet. Vous pouvez ajouter des méthodes et des propriétés personnalisées à des formulaires et y accéder à partir d'autres formulaires ou d'autres classes de votre application.

Pour créer une nouvelle méthode pour un formulaire

- Ajoutez une procédure déclarée **Public**, comme dans le code suivant :

  VB

```
' Create a custom method on a form.
Public Sub PrintMyJob()
    ' Insert the code for your method here.
End Sub
```

Pour ajouter un nouveau champ à un formulaire

- Déclarez une variable publique dans le module de formulaire, comme dans le code suivant :

  VB

```
Public IDNumber As Integer
```

Pour accéder aux méthodes d'un autre formulaire

1. Créez une instance du formulaire aux méthodes duquel vous voulez accéder. Lorsque vous référencez un nom de formulaire, vous référencez en réalité la classe à laquelle appartient ce formulaire et non l'objet lui-même.

   Remarque

   Visual Basic fournit une variable globale implicite du même nom que le formulaire pour chaque classe Form. Pour plus d'informations, consultez Comment : accéder à un formulaire.

2. Assignez le formulaire à une variable d'objet. La variable d'objet référence une nouvelle instance de la classe Form.

   L'exemple suivant appelle correctement la procédure `PrintMyJob` :

   VB

```
Dim newForm1 As New Form1
newForm1.PrintMyJob()
```

   Dans l'exemple précédent, le nouveau formulaire n'est pas affiché. Il n'est pas nécessaire d'afficher un objet de formulaire pour utiliser ses méthodes. Pour afficher le nouveau formulaire, vous devez ajouter le code suivant :

   VB

```
newForm1.Show()
```

Voir aussi

Tâches

Comment : utiliser le nouveau mot clé

Concepts

Procédures Property

Comment : accéder à un formulaire

Vous pouvez accéder aux membres d'un formulaire Visual Basic sans avoir à créer de variable. Les exemples suivants démontrent cela en modifiant des couleurs sur un formulaire.

Accès à un formulaire

Pour accéder à Form1

1. Assurez-vous que votre projet a une référence à l'espace de noms System.Drawing. Cela est nécessaire pour travailler avec les paramètres de couleur, pas pour accéder au formulaire.
2. Modifiez directement les couleurs sur `Form1`.
3. Appelez directement la méthode Show sur `Form1`.

   VB

   ```
   Public Sub ChangeForm1Colors()
       Form1.ForeColor = System.Drawing.Color.Coral
       Form1.BackColor = System.Drawing.Color.Cyan
       Form1.Show()
   End Sub
   ```

   Si `Form1` n'existe pas déjà, Visual Basic le crée pour vous. Vous n'êtes pas obligé de déclarer une variable pour lui.

Création d'une instance supplémentaire d'un formulaire

Si vous souhaitez créer un nouveau formulaire, plutôt qu'accéder à un formulaire existant, vous pouvez déclarer une variable et l'initialiser à l'aide du mot clé **New**.

Pour créer une copie supplémentaire de Form1

1. Assurez-vous que votre projet a une référence à l'espace de noms **System.Drawing**. Cela est nécessaire pour travailler avec les paramètres de couleur, pas pour accéder au formulaire.
2. Assignez `New Form1` à une variable.

   VB

   ```
   Public Sub GetSecondInstance()
       Dim newForm1 As New Form1
       newForm1.BackColor = System.Drawing.Color.YellowGreen
       newForm1.Show()
   End Sub
   ```

   Si vous souhaitez afficher plusieurs copies du même formulaire, vous devez créer les copies supplémentaires. L'exemple précédent crée une deuxième copie de `Form1` et la colorie avec une couleur différente. Vous pouvez accéder ensuite à la copie d'origine à l'aide de `Form1` et à la deuxième copie à l'aide de `newForm1`.

Voir aussi

Référence

New (Visual Basic)
System.Drawing
Form

Comment : utiliser le nouveau mot clé

Pour créer une instance d'une classe, utilisez le mot clé **New**. À la différence de **Integer** et **Double** qui sont des types valeur, les objets sont des types référence, et vous devez donc les créer explicitement avant de pouvoir les utiliser. Considérons par exemple les deux lignes de code suivantes :

VB

```
Dim Button1 As System.Windows.Forms.Button
Dim Button2 As New System.Windows.Forms.Button()
```

La première instruction déclare une variable d'objet qui peut contenir une référence à un objet Button. Toutefois, la variable `Button1` contient la valeur **Nothing** jusqu'à ce que vous lui assigniez un objet de type Button. La seconde instruction définit également une variable pouvant contenir un objet Button, mais le mot clé **New** crée un objet Button et l'assigne à la variable `Button2`.

Les formulaires et les contrôles étant en réalité des classes, vous pouvez utiliser le mot clé **New** pour créer de nouvelles instances de ces éléments en fonction de vos besoins.

Pour créer de nouvelles instances d'une classe à l'aide du mot clé New

1. Ouvrez un nouveau projet Application Windows, puis placez un bouton de commande et plusieurs autres contrôles dans un formulaire appelé `Form1`.
2. Ajoutez le code suivant à la procédure événementielle Click du bouton de commande :

   VB

   ```
   Dim f As New Form1
   f.Show()
   ```

3. Exécutez l'application et cliquez plusieurs fois sur le bouton de commande.
4. Mettez de côté le formulaire de premier plan. Étant donné qu'un formulaire est une classe dotée d'une interface visible, vous pouvez voir les autres copies. Chacune de ces copies contient les mêmes contrôles et aux mêmes positions que ceux du formulaire d'origine au moment du design.

Vous pouvez utiliser le mot clé **New** pour créer des objets à partir des classes. La procédure suivante vous montre comment.

Pour voir comment le mot clé New crée des instances d'une classe

1. Ouvrez un nouveau projet Windows Forms, puis placez un bouton de commande dans un formulaire appelé `Form1`.
2. Dans le menu **Projet**, sélectionnez **Ajouter une classe** pour ajouter une classe au projet.
3. Donnez à la classe le nom `ShowMe.vb`.
4. Ajoutez la procédure suivante à `ShowMe`:

   VB

   ```
   Public Class ShowMe
       Sub ShowFrm()
           Dim frmNew As Form1
           frmNew = New Form1
           frmNew.Show()
           frmNew.WindowState = FormWindowState.Minimized
       End Sub
   End Class
   ```

5. Ajoutez le code suivant pour gérer l'événement **Click** de `Button1` sur votre formulaire :

   VB

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
    Dim clsNew As New ShowMe
    clsNew.ShowFrm()
End Sub
```

6. Pour utiliser l'exemple, exécutez l'application et cliquez plusieurs fois sur le bouton de commande. Une icône de formulaire réduite apparaît dans votre barre des tâches chaque fois qu'est créée une nouvelle instance de la classe `ShowMe`.

Voir aussi

Autres ressources

Création et utilisation d'objets

Gestion des ressources

Tous les objets consomment des ressources système, comme de la mémoire, des handles de fichier et des connexions de base de données. Le Common Language Runtime (CLR) gère automatiquement les ressources ; vous ne devez généralement pas vous préoccuper de libérer des objets inutiles. Toutefois, la compréhension du fonctionnement de la gestion des ressources peut vous aider à concevoir plus efficacement vos applications.

Garbage collection

Le CLR utilise un système appelé garbage collection pour gérer les ressources allouées. Le garbage collector (également appelé ramasse-miettes) libère les ressources d'un objet lorsque le code d'exécution de votre application ne peut plus atteindre cet objet. L'algorithme du garbage collection n'est pas déterministe. De ce fait, vous ne pouvez pas déterminer à quel moment le CLR libère les ressources d'un objet. Les sections suivantes décrivent certaines modifications de la gestion des ressources en Visual Basic.

Assignation d'objets à Nothing

Nothing est un mot clé utilisé par Visual Basic pour indiquer qu'une variable objet ne contient pas de référence à un objet. Les versions antérieures de Visual Basic vous encourageaient à assigner des objets non utilisés à **Nothing** pour dissocier la variable d'objet de l'objet et libérer des ressources. Vous pouvez toujours assigner des objets non utilisés à **Nothing** mais, en raison de la façon dont Visual Basic gère les ressources, ce processus ne garantit pas une libération immédiate des objets. En règle générale, vous devez uniquement assigner des objets à longue durée de vie, tels que des membres partagés ou des variables globales, à **Nothing**.

Supprimer

Certains objets prennent en charge la méthode Dispose dont le but est de libérer plus rapidement des ressources système. Les classes qui prennent en charge la méthode **Dispose** doivent implémenter l'interface IDisposable. La méthode **Dispose** doit être appelée explicitement lorsque vous voulez libérer des ressources d'objet. Par exemple :

ThisObject.**Dispose**()

Finalize

La méthode Finalize, prise en charge par certaines classes, s'exécute automatiquement lorsqu'un objet est libéré et peut être utilisée pour effectuer d'autres tâches de nettoyage. La méthode **Finalize** est semblable à la méthode **Class_Terminate()** utilisée dans les versions précédentes de Visual Basic. Une fois qu'un objet devient inaccessible, le CLR appellera finalement la méthode **Finalize** de l'objet. Étant donné que l'algorithme du garbage collection n'est pas déterministe, la méthode **Finalize** peut être appelée immédiatement ou plusieurs heures après.

Voir aussi

Référence

Nothing (Visual Basic)
Dispose
IDisposable

Concepts

Durée de vie d'un objet : création et destruction des objets
Initialisation et suppression des composants

Comment : passer des objets aux procédures

Visual Basic vous permet de passer des objets en tant qu'arguments à des procédures comme vous passez d'autres types d'arguments. Les procédures suivantes vous montrent comment.

Pour passer une nouvelle instance d'un formulaire à une procédure

1. Ouvrez un projet, créez un formulaire appelé `Form1` et ajoutez un bouton de commande appelé `Button1`.
2. Copiez le code suivant dans le formulaire :

   VB

```
Private Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    Dim newForm As New Form1
    newForm.Show()
    CenterForm(newForm)
End Sub

Sub CenterForm(ByVal TheForm As Form)
    ' Centers the form on the screen.
    Dim RecForm As Rectangle = Screen.GetBounds(TheForm)
    TheForm.Left = CInt((RecForm.Width - TheForm.Width) / 2)
    TheForm.Top = CInt((RecForm.Height - TheForm.Height) / 2)
End Sub
```

   Vous pouvez également passer un objet comme argument par référence puis, dans la procédure, définir l'argument comme un nouvel objet.

Pour passer une référence d'objet à une procédure d'un autre formulaire

1. Ouvrez un projet et créez un formulaire appelé `Form1`.
2. Ajoutez un second formulaire appelé `Form2`.
3. Placez un contrôle de zone d'image dans chaque formulaire.
4. Donnez à la zone d'image du formulaire Form1 le nom `PictureBox1`.
5. Donnez à la zone d'image du formulaire Form2 le nom `PictureBox2`.
6. Assignez une image à `PictureBox2` en cliquant sur la propriété **Image** dans la fenêtre **Propriétés**. Vous pouvez utiliser n'importe quelle petite image ; vous trouverez des fichiers .bmp et .jpg dans votre répertoire Windows.
7. Ajoutez le code suivant à `Form2`:

   VB

```
Public Sub GetPicture(ByVal x As PictureBox)
    Dim objX As PictureBox
    ' Assign the passed-in picture box to an object variable.
    objX = x
    ' Assign the value of the Picture property to the Form1 picture box.
    objX.Image = PictureBox2.Image
End Sub
```

8. Ajoutez le code suivant à l'événement `Form1_Click` sur `Form1` :

   VB

```
Protected Sub Form1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs)
    Dim newForm2 As New Form2
    newForm2.GetPicture(PictureBox1)
End Sub
```

9. Exécutez l'application et cliquez sur `Form1`. L'image de `Form2` apparaît dans la zone d'image de `Form1`.

   La procédure événementielle `Form1_Click` appelle la procédure `GetPicture` dans `Form2` et lui passe la zone d'image vide. La procédure `GetPicture` de `Form2` assigne la propriété **Image** de la zone d'image de `Form2` à la zone d'image vide de `Form1`, et l'image de `Form2` est affichée dans `Form1`.

Voir aussi

Concepts

Gestion des ressources

Autres ressources

Programmation orientée objet dans Visual Basic

Programmation à l'aide de composants

Procédure pas à pas : persistance d'un objet en Visual Basic

Bien qu'il soit possible de définir des valeur par défaut pour les propriétés d'un objet au moment du design, les valeurs entrées lors de l'exécution sont perdues en cas de destruction de l'objet. Vous pouvez utiliser la sérialisation en Visual Basic pour rendre les données d'un objet persistantes entre les instances, ce qui vous permet de stocker des valeurs et de les récupérer lors de la prochaine instanciation de l'objet.

Lorsque vous stockez des données simples, telles qu'un nom ou un nombre, vous pouvez utiliser l'objet **My.Settings**. Pour plus d'informations, consultez My.Settings, objet.

Dans cette procédure pas à pas, vous allez créer un objet `Loan` simple et rendre ses données persistantes dans un fichier, puis récupérer les données du fichier lors de la recréation de l'objet. Vous allez ensuite modifier le code afin de rendre l'objet persistant dans un format SOAP.

Remarque de sécurité

Cet exemple crée un fichier s'il n'existe pas. Si une application doit créer un fichier, elle doit disposer de l'autorisation **Create** pour accéder au dossier. Les autorisations sont définies à l'aide de listes de contrôle d'accès. Si le fichier existe déjà, l'application a uniquement besoin de l'autorisation **Write**, ce qui représente un privilège inférieur. Lorsque cela est possible, il est plus sûr de créer le fichier au cours du déploiement et de n'accorder les privilèges **Read** qu'à un seul fichier (plutôt que d'accorder des privilèges Créer à un dossier). En outre, il est plus sûr d'écrire les données dans des dossiers utilisateur que dans le dossier racine ou le dossier Program Files.

Remarque de sécurité

Cet exemple stocke les données dans un fichier au format binaire ou SOAP. Ces formats ne doivent pas être utilisés pour données sensibles, telles que les mots de passe ou les informations relatives à la carte de crédit.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Création de l'objet Loan

La première étape consiste à créer une classe `Loan` et une application de test qui utilise cette classe.

Pour créer la classe Loan

1. Créez un projet Bibliothèque de classes et nommez-le "LoanClass". Pour plus d'informations, consultez Comment : créer des solutions et des projets.
2. Dans l'**éditeur de code**, remplacez le nom de la classe "Class1" par "Loan".
3. Ajoutez les membres publics suivants à la classe :

   VB

   ```
   Public LoanAmount As Double = 10000.0
   Public InterestRate As Double = 7.5
   Public Term As Integer = 36
   Public Customer As String
   ```

Vous devrez également créer une application simple qui utilise la classe `Loan`.

Pour créer une application de test

1. Dans le menu **Fichier**, ajoutez un projet d'application Windows à votre solution en pointant sur **Ajouter** et en cliquant ensuite sur **Nouveau projet**.

2. Dans la boîte de dialogue **Ajouter un nouveau projet**, entrez **LoanApp** comme nom du projet et cliquez sur **OK** pour fermer la boîte de dialogue.
3. Dans l'**Explorateur de solutions**, sélectionnez le projet LoanApp.
4. Dans le menu **Projet**, cliquez sur **Définir comme projet de démarrage**.
5. Dans le menu **Projet**, cliquez sur **Ajouter une référence**.
6. Dans la boîte de dialogue **Ajouter une référence**, cliquez sur l'onglet **Projets**, puis sur le projet LoanClass.
7. Cliquez sur **OK** pour fermer la boîte de dialogue.
8. Dans le concepteur, ajoutez quatre contrôles TextBox au formulaire.
9. Dans l'éditeur de code, ajoutez le code suivant :

   VB

```
Private TestLoan As New LoanClass.Loan
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    TextBox1.Text = TestLoan.LoanAmount.ToString
    TextBox2.Text = TestLoan.InterestRate.ToString
    TextBox3.Text = TestLoan.Term.ToString
    TextBox4.Text = TestLoan.Customer
End Sub
```

À ce stade, vous pouvez générer et exécuter l'application. Remarquez que les valeurs par défaut de la classe `Loan` apparaissent dans les zones de texte. Essayez de remplacer le taux d'intérêt de 7,5 par 7,1, puis fermez l'application et exécutez-la de nouveau : le taux reprend sa valeur par défaut de 7,5.

Dans la réalité, les taux d'intérêt changent régulièrement, mais pas nécessairement chaque fois que l'application est exécutée. Plutôt que d'imposer à l'utilisateur de mettre à jour le taux d'intérêt chaque fois qu'il exécute l'application, il serait plus judicieux de conserver le taux d'intérêt le plus récent entre les instances de l'application. C'est ce que vous ferez à l'étape suivante, en ajoutant la sérialisation à la classe Loan.

Utilisation de la sérialisation pour rendre l'objet persistant

Pour rendre les valeurs de la classe Loan persistantes, vous devez d'abord marquer la classe avec l'attribut **Serializable**.

Pour marquer une classe comme étant sérialisable

- Modifiez la déclaration de la classe Loan de la façon suivante :

  VB

```
<Serializable()> Public Class Loan
```

L'attribut **Serializable** indique au compilateur que tout ce que contient la classe peut être persistant dans un fichier. Dans ce cas, vous souhaitez conserver uniquement le membre `InterestRate`, et non les membres `Customer`, `LoanAmount` ou `Period`. L'attribut **NonSerialized** peut être utilisé pour marquer les membres de la classe qui ne doivent pas être persistants. Pour simplifier cet exemple, tout est rendu persistant, sauf le membre `Customer`.

Pour empêcher la sérialisation d'un membre

- Modifiez la déclaration du membre `Customer` de la façon suivante :

  VB

```
<NonSerialized()> Public Customer As String
```

L'étape suivante consiste à ajouter le code de sérialisation à l'application LoanApp. Afin de sérialiser la classe et de l'écrire dans

un fichier, utilisez les noms d'espaces System.IO et System.Xml.Serialization. Pour éviter d'avoir à saisir les noms qualifiés complets, vous pouvez utiliser l'instruction **Imports**.

Pour ajouter des références aux espaces de noms

- Ajoutez les instructions **Imports** suivantes au début de la classe `Form1` :

  VB

  ```
  Imports System.IO
  Imports System.Runtime.Serialization.Formatters.Binary
  ```

  Dans ce cas, vous utilisez un formateur binaire pour enregistrer l'objet dans un format binaire. Plus loin dans cette procédure, vous allez modifier le code afin d'enregistrer l'objet dans un format SOAP.

L'étape suivante consiste à ajouter le code permettant de désérialiser l'objet à partir du fichier lors de la création de l'objet.

Pour désérialiser un objet

1. Ajoutez une constante à la classe pour le nom de fichier des données sérialisées.

   VB

   ```
   Const FileName As String = "SavedLoan.bin"
   ```

2. Modifiez le code de la procédure événementielle `Form1_Load` de la façon suivante :

   VB

   ```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
   System.EventArgs) Handles MyBase.Load
       If File.Exists(FileName) Then
           Dim TestFileStream As Stream = File.OpenRead(FileName)
           Dim deserializer As New BinaryFormatter
           TestLoan = CType(deserializer.Deserialize(TestFileStream), LoanClass.Loan)
           TestFileStream.Close()
       End If
       TextBox1.Text = TestLoan.LoanAmount.ToString
       TextBox2.Text = TestLoan.InterestRate.ToString
       TextBox3.Text = TestLoan.Term.ToString
       TextBox4.Text = TestLoan.Customer
   End Sub
   ```

   Remarquez que vous devez d'abord vérifier que le fichier existe. S'il existe, créez une classe Stream pour lire le fichier binaire et une classe BinaryFormatter pour convertir le fichier. La méthode **CType** est utilisée pour la conversion du type d'objet Stream en type Loan.

Vous devez ensuite ajouter le code permettant d'enregistrer dans la classe `Loan` les données saisies dans les zones de texte, puis ensuite de sérialiser la classe dans un fichier.

Pour enregistrer les données et sérialiser la classe

- Ajoutez le code suivant à la procédure événementielle `Form1_Closing` :

  VB

  ```
  Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As _
  System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
      TestLoan.LoanAmount = CType(TextBox1.Text, Double)
      TestLoan.InterestRate = CType(TextBox2.Text, Double)
  ```

```
    TestLoan.Term = CType(TextBox3.Text, Integer)
    TestLoan.Customer = TextBox4.Text

    Dim TestFileStream As Stream = File.Create(FileName)
    Dim serializer As New BinaryFormatter
    serializer.Serialize(TestFileStream, TestLoan)
    TestFileStream.Close()
End Sub
```

À ce stade, vous pouvez de nouveau générer et exécuter l'application. À l'origine, les valeurs par défaut s'affichent dans les zones de texte. Essayez de modifier les valeurs et de saisir un nom dans la quatrième zone de texte. Fermez l'application et exécutez-la de nouveau. Remarquez que les nouvelles valeurs s'affichent à présent dans les zones de texte, excepté le nom du client, qui a été marqué comme **NonSerialized**.

Persistance de l'objet à l'aide d'un format SOAP

Jusqu'à présent, l'exemple a illustré la persistance d'un objet dans un fichier texte à l'aide d'un format binaire. Un tel format est parfait pour la plupart des applications Windows. En revanche, pour les applications Web ou les services Web XML, vous souhaiterez peut-être rendre l'objet persistant dans un fichier XML à l'aide d'un format SOAP, ce qui facilite le partage de l'objet.

Pour rendre l'objet persistant dans un format SOAP, vous devez d'abord référencer la classe SoapFormatter. La classe **SoapFormatter** se trouve dans son propre espace de noms : System.Runtime.Serialization.Formatters.Soap.

Pour rendre persistant l'objet à l'aide d'un format SOAP

1. Dans l'**Explorateur de solutions**, sélectionnez le projet LoanApp.
2. Dans le menu **Projet**, cliquez sur **Ajouter une référence**.
3. Dans la boîte de dialogue **Ajouter une référence**, cliquez sur l'onglet **.NET**, puis sur le composant **System.Runtime.Serialization.Formatters.Soap**.
4. Cliquez sur **OK** pour fermer la boîte de dialogue.
5. Dans l'**éditeur de code**, ajoutez une instruction **Imports** au début du module `Form1` :

   VB
   ```
   Imports System.Runtime.Serialization.Formatters.Soap
   ```
6. Remplacez le nom de fichier `SavedLoan.bin` par `SavedLoan.xml`.
7. Dans la procédure événementielle `Form1_Load`, modifiez l'instruction **Dim** de `Dim deserializer As New BinaryFormatter` en :

   VB
   ```
   Dim deserializer As New SoapFormatter
   ```
8. Dans la procédure événementielle `Form1_Closing`, modifiez l'instruction **Dim** de `Dim serializer As New BinaryFormatter` en :

   VB
   ```
   Dim serializer As New SoapFormatter
   ```

À ce stade, vous pouvez générer et tester l'application. Lors de la première exécution de l'application, le fichier SavedLoan.xml est créé. Vous pouvez afficher ce fichier en sélectionnant l'option **Afficher tous les fichiers** dans l'**Explorateur de solutions** ; il se trouve sous le nœud Bin du projet d'application Windows.

Remarque

Si le mode **Afficher tous les fichiers** est déjà activé, vous devez actualiser la vue en choisissant **Actualiser** dans le menu **Affichage** pour voir le fichier.

Remarquez que les trois membres de `LoanClass` s'affichent au format XML. Essayez de modifier la valeur InterestRate dans le fichier XML, puis enregistrez-le et exécutez de nouveau l'application. Le nouveau taux d'intérêt s'affiche dans la deuxième zone de texte.

Voir aussi

Concepts

Équivalents PropertyBag pour les utilisateurs de Visual Basic 6.0

Concepts de base de la sérialisation .NET Framework

Comment : hériter d'une classe dans Visual Basic

Cet exemple définit les classes `Circle` et `Rectangle` (qui héritent toutes deux de la classe `Shape`) et la classe `Square` (qui hérite de la classe `Rectangle`).

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
Public Class Shape
    ' Definitions of properties, methods, fields, and events.
End Class
Public Class Circle : Inherits Shape
    ' Specialized properties, methods, fields, events for Circle.
End Class
Public Class Rectangle : Inherits Shape
    ' Specialized properties, methods, fields, events for Rectangle.
End Class
Public Class Square : Inherits Rectangle
    ' Specialized properties, methods, fields, events for Square.
End Class
```

Compilation du code

Cet exemple nécessite :

- Référence à l'espace de noms System.

Remarque
Assurez-vous que la classe à partir de laquelle vous souhaitez hériter n'est pas définie comme **NotInheritable**.

Voir aussi

Tâches

Comment : dessiner une forme avec contour

Référence

NotInheritable

Autres ressources

Héritage en Visual Basic

Création et utilisation d'objets

Comment : lire des données d'objet à partir d'un fichier XML dans Visual Basic

Cet exemple lit des données d'objet écrites précédemment dans un fichier XML à l'aide de la classe XmlSerializer.

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **XML**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
Public Class Book
    Public Title As String
    Public Sub New()
    End Sub
End Class

Public Sub ReadXML()
    Dim reader As New System.Xml.Serialization.XmlSerializer(GetType(Book))
    Dim file As New System.IO.StreamReader("c:\IntroToVB.xml")
    Dim introToVB As Book
    introToVB = CType(reader.Deserialize(file), Book)
End Sub
```

Compilation du code

Remplacez le nom de fichier "c:\IntroToVB.xml" par le nom du fichier contenant les données sérialisées. Pour plus d'informations sur la sérialisation des données, consultez
Comment : écrire des données d'objet dans un fichier XML en Visual Basic.

La classe doit disposer d'un constructeur public sans paramètres.

Seuls les propriétés et les champs publics sont désérialisés.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception.

- La classe en cours de désérialisation ne dispose pas d'un constructeur public sans paramètres.
- Les données contenues dans le fichier ne correspondent pas aux données de la classe à désérialiser.
- Le fichier n'existe pas (IOException).

Sécurité

Vérifiez toujours les entrées et ne désérialisez jamais des données à partir d'une source non fiable. L'objet recréé s'exécute sur un ordinateur local avec les autorisations du code qui l'a désérialisé. Vérifiez toutes les entrées avant d'utiliser les données dans votre application.

Voir aussi

Tâches

Comment : écrire des données d'objet dans un fichier XML en Visual Basic
Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

Référence

StreamWriter

Comment : écrire des données d'objet dans un fichier XML en Visual Basic

Cet exemple écrit l'objet d'une classe dans un fichier XML à l'aide de la classe XmlSerializer.

Exemple

Cet exemple de code définit une classe nommée `Book`, crée une instance de la classe et utilise la sérialisation XML pour écrire l'instance dans un fichier XML.

Du code semblable à celui-ci est disponible sous la forme d'un extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve dans **XML**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
Public Class Book
    Public Title As String
    Public Sub New()
    End Sub
End Class

Public Sub WriteXML()
    Dim introToVB As New Book
    introToVB.Title = "Intro to Visual Basic"
    Dim writer As New System.Xml.Serialization.XmlSerializer(GetType(Book))
    Dim file As New System.IO.StreamWriter("c:\IntroToVB.xml")
    writer.Serialize(file, introToVB)
    file.Close()
End Sub
```

Compilation du code

La classe doit disposer d'un constructeur public sans paramètres.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception :

- La classe en cours de désérialisation ne dispose pas d'un constructeur public sans paramètres.
- Le fichier existe déjà et est en lecture seule (IOException).
- Le chemin d'accès est trop long (PathTooLongException
- Le disque est saturé (**IOException**).

Sécurité

Cet exemple crée un fichier s'il n'existe pas. Si une application doit créer un fichier, elle doit disposer de l'accès **Create** pour le dossier. Si le fichier existe déjà, l'application n'a besoin que d'un accès **Write**, un privilège moins important. Dans la mesure du possible, et pour des raisons de sécurité, il est préférable de créer le fichier au moment du déploiement et d'accorder uniquement un accès **Read** à un seul fichier plutôt qu'un accès **Create** à l'ensemble du dossier.

Voir aussi

Tâches

Comment : lire des données d'objet à partir d'un fichier XML dans Visual Basic

Référence

StreamWriter

Autres ressources

Accès au fichier avec Visual Basic

Collections en Visual Basic

En règle générale, une *collection* est un objet utilisé pour grouper et gérer des objets apparentés. Par exemple, chaque Form a une collection de contrôles. (Vous pouvez accéder à cette collection par l'intermédiaire de la propriété Controls du formulaire). Cette collection est un objet qui représente tous les contrôles de ce formulaire. Elle permet de récupérer un contrôle de la collection en fonction de son index et de parcourir les éléments de la collection à l'aide de For Each...Next, instruction (Visual Basic).

Toutefois, il existe plusieurs types de collections qui diffèrent les unes des autres de plusieurs façons.

Différents types de collection

Visual Basic fournit également une classe **Collection**, qui vous permet de définir et de créer vos propres collections. À l'instar de la collection **Controls** d'un formulaire, la classe **Collection** offre également une fonctionnalité intégrée qui vous permet de parcourir les éléments à l'aide de **For Each**...**Next** et de récupérer des éléments en fonction de leur index. Pour plus d'informations, consultez Collection, objet (Visual Basic).

Toutefois, les deux types de collection n'interagissent pas entre eux. Par exemple, le code suivant génère une erreur de compilation :

```
Dim localControls As Collection

' The following line generates a COMPILER ERROR.

localControls = Me.Controls()
```

Les collections sont incompatibles car la collection **Controls** est une collection du .NET Framework, tandis que la variable `localControls` est une **Collection** de Visual Basic. Les deux types de collections sont implémentés à partir de classes différentes. Leurs méthodes sont semblables mais pas identiques, et leurs modèles d'indexation sont différents.

Collections de base zéro et de base un

Une collection peut être soit de *base zéro*, soit de *base un*, selon son index de départ. Une collection de base zéro signifie que l'index du premier élément de la collection est 0, et une collection de base un signifie que l'index est 1. Un exemple de collection de base zéro est la collection **Controls** du .NET Framework, décrite précédemment dans cette page. L'objet **Collection** de Visual Basic est un exemple de collection de base un.

Les collections de base un peuvent être plus intuitives aux utilisateurs Visual Basic, car l'index est compris entre 1 et la valeur de Count, propriété (objet Collection), qui retourne le nombre d'éléments contenus dans une collection. En revanche, l'index d'une collection de base zéro est compris entre 0 et une valeur inférieure d'une unité à la valeur de la propriété **Count**. Cela peut être approprié lorsque les valeurs d'index représentent des offsets d'une valeur de base ou correspondent aux membres d'une énumération de base zéro.

Les collections du .NET Framework sont de base zéro pour les besoins de normalisation. La classe **Collection** de Visual Basic est une classe de base un qui sert à assurer une compatibilité avec des versions précédentes.

Valeurs d'index et de clés

Les instances de la classe **Collection** de Visual Basic vous permettent d'accéder à un élément à l'aide d'un index numérique ou d'une clé **String**. Vous pouvez ajouter des éléments aux objets **Collection** de Visual Basic en spécifiant ou non une clé. Si vous ajoutez un élément sans clé, vous devez utiliser son index numérique pour y accéder.

En revanche, les collections telles que System.Collections.ArrayList autorisent uniquement un index numérique. Vous ne pouvez pas associer de clés aux éléments de ces collections, à moins que vous construisiez votre propre mappage basé, par exemple, sur un tableau **String** qui contient les clés.

Ajout et suppression d'éléments

Une autre différence entre les collections est la possibilité d'y ajouter des éléments ainsi que la façon dont ces éléments sont ajoutés, le cas échéant. Puisque l'objet **Collection** de Visual Basic est un outil de programmation à caractère général, il est plus flexible que certaines autres collections. Il dispose de Add, méthode (objet Collection) pour insérer des éléments dans la collection et de Remove, méthode (objet Collection) pour récupérer des éléments.

Par contre, certaines collections spécialisées ne vous permettent pas d'ajouter ou de supprimer des éléments en utilisant du code. Par exemple, la propriété System.Windows.Forms.CheckedListBox.CheckedItems retourne une collection de références à des éléments en fonction de leur index, mais votre code ne peut pas ajouter ou supprimer des éléments de cette collection. Seul l'utilisateur peut le faire en sélectionnant ou en effaçant la zone appropriée dans l'interface utilisateur. En effet, cette

collection ne possède pas de méthode **Add** ou **Remove**.

Voir aussi

Tâches

Comment : créer une collection d'objets
Comment : créer un tableau d'objets
Comment : ajouter, supprimer et récupérer des éléments d'une collection
Comment : définir des collections dans vos classes
Comment : parcourir une collection dans Visual Basic
Dépannage des collections

Concepts

Gestion de groupes d'objets
Classe de la collection Visual Basic
Gestion de vos objets avec les collections

Gestion de groupes d'objets

Pour de nombreuses applications, vous voulez être en mesure de créer et de gérer des groupes d'objets apparentés. Il existe deux manières de grouper des objets : en créant des tableaux d'objets ou des collections d'objets.

Tableaux d'objets

Les tableaux sont des structures assez rigides. Si vous souhaitez modifier la taille du tableau au moment de l'exécution, vous devez utiliser l'ReDim, instruction (Visual Basic) pour le redéclarer. Tous les éléments du tableau doivent être du même type. En revanche, les tableaux vous permettent de traiter tous les éléments les uns après les autres et de posséder des éléments vides. Les tableaux sont donc plus utiles pour la création et l'utilisation d'un nombre fixe d'objets fortement typés.

Pour plus d'informations, consultez Comment : créer un tableau d'objets.

Collections d'objets

Les collections offrent plus de souplesse pour utiliser des groupes d'objets. Dans la mesure où une collection est une classe, vous devez déclarer une nouvelle collection avant de pouvoir lui ajouter des éléments. Contrairement aux tableaux, le groupe d'objets utilisé peut augmenter ou diminuer de façon dynamique si les besoins de l'application varient. Vous pouvez assigner une *clé* à un objet que vous insérez dans une collection afin de pourvoir l'extraire et le manipuler en fonction de cette clé.

Vous pouvez créer une collection généralisée à partir de la classe Visual Basic Collection. Si vous souhaitez disposer d'une fonctionnalité spécifique dans une collection, par exemple un dictionnaire ou une liste liée, vous pouvez la créer à partir d'une des classes de l'espace de noms System.Collections du .NET Framework. Même les classes de collection plus spécialisées sont disponibles dans l'espace de noms System.Collections.Specialized.

Si votre collection se limite à des éléments d'un seul type de données, vous pouvez utiliser l'une des classes de l'espace de noms System.Collections.Generic. Une collection *générique* met en vigueur la *sécurité de type* afin qu'aucun autre type de données ne puisse lui être ajouté. Lorsque vous récupérez un élément à partir d'une collection générique, il n'est pas nécessaire de déterminer son type de données ni de le convertir.

Pour plus d'informations, consultez Comment : créer une collection d'objets.

Voir aussi

Tâches

Comment : ajouter, supprimer et récupérer des éléments d'une collection

Référence

System.Collections
System.Collections.Generic
System.Collections.Specialized

Concepts

Collections en Visual Basic
Classe de la collection Visual Basic

Autres ressources

Tableaux dans Visual Basic
Création et utilisation d'objets

Gestion de vos objets avec les collections

Les collections vous permettent de gérer efficacement un ensemble d'objets. Vous pouvez ajouter et supprimer des objets d'une collection, les extraire en fonction d'un index ou d'une clé et utiliser For Each...Next, instruction (Visual Basic) pour parcourir les éléments de votre collection.

Collections de type risquées

Toutefois, la flexibilité des collections peut fragiliser la robustesse de vos classes. Par exemple, la collection fournie par Visual Basic stocke tous ses éléments comme type **Object**, vous pouvez donc ajouter un élément de n'importe quel type de données. Il n'existe aucun dispositif de protection contre les types de données peu appropriés qui sont ajoutés, et lorsque vous accédez à un élément, vous devez le convertir de **Object** en type de données voulu.

Collections spécialisées

Le .NET Framework fournit plusieurs alternatives à la collection Visual Basic. L'espace de noms System.Collections contient des classes de collection avec des fonctionnalités spécifiques, telles qu'une file d'attente ou une liste triée, et l'espace de noms System.Collections.Specialized contient des classes de collection particulière, telles que HybridDictionary.

Collections de type sécurisées

Pour éviter les inconvénients d'éléments de type **Object**, vous pouvez utiliser les collections *génériques* de l'espace de noms System.Collections.Generic. Ces collections fournissent la *sécurité de type* et vous permettent de limiter les éléments d'une collection à un seul type de données spécifique.

Approches de l'utilisation des collections

Vous pouvez suivre trois approches générales pour implémenter une gestion d'objets en utilisant des collections. Considérez une application qui définit une classe `widgetRepository` qui organise et expose des objets `widget` à des composants clients. Pour implémenter `widgetRepository` avec une collection, vous pouvez utiliser l'une des stratégies suivantes.

- **Utiliser une classe de collection.** Dans la classe `widgetRepository`, déclarez une variable `widgetsColl` comme une instance de Classe de la collection Visual Basic ou de l'une des classes dans les espaces de noms **System.Collections**, **System.Collections.Generic** ou **System.Collections.Specialized**. Rendez la variable publique et utilisez le mot clé New (Visual Basic) pour créer une instance de la collection. Pour plus d'informations, consultez Comment : définir des collections dans vos classes.
- **Hériter d'une classe de base de collection.** Implémentez votre propre classe `widgetsColl` en héritant de la classe CollectionBase. Dans la classe `widgetRepository`, définissez une instance de la classe `widgetsColl` et une propriété qui retourne cette instance. Pour plus d'informations, consultez Comment : définir des collections dans vos classes.
- **Écrire la collection vous-même.** Implémentez les fonctionnalités de la collection dans la classe `widgetRepository` en écrivant les classes et les procédures appropriées. Cette approche est très utile si vous avez besoin de fonctionnalités de collection dans votre classe, mais ne pouvez hériter des classes de collection existantes. Cela est possible, par exemple, si votre classe devait, exceptionnellement, hériter d'une classe autre qu'une classe de collection. Comme elle ne peut pas hériter de plusieurs classes, vous devrez définir et implémenter des membres de collection.

Voir aussi

Référence

System.Collections
System.Collections.Generic
System.Collections.Specialized

Concepts

Gestion de groupes d'objets
Classe de la collection Visual Basic

Classe de la collection Visual Basic

Une *collection* permet de grouper un ensemble d'éléments apparentés. Il existe plusieurs types de collection. Les collections prédéfinies sont utilisées dans les applications Visual Basic à des fins multiples, par exemple le ControlCollection sur un Form, retourné par la propriété Controls du formulaire. Vous pouvez également créer vos propres collections pour organiser et manipuler des objets.

Les collections représentent une méthode efficace pour assurer le suivi des objets que votre application doit créer et détruire de façon dynamique. Le fragment de code suivant illustre la façon d'utiliser la méthode Add Method d'un objet Collection Visual Basic afin de conserver une liste des objets `widget` créés par l'utilisateur.

```
' Declare and create the Collection object.
Public widgetColl As New Microsoft.VisualBasic.Collection()
' Create a new widget and add it to the widgetColl collection.
Private Sub makeAWidget()
    Dim tempWidget As New widget()
    widgetColl.Add(tempWidget)
End Sub
```

Dans l'exemple précédent, la collection `widgetColl` organise et expose tous les objets `widget` créés par l'intermédiaire de la procédure `makeAWidget`. Vous pouvez extraire des références d'objet de chaque objet `widget` par l'index de la collection. La taille de la collection est automatiquement ajustée lorsqu'un nouvel objet `widget` est ajouté. Les instructions For Each...Next, instruction (Visual Basic) vous permettent de parcourir la collection. Si vous souhaitez assigner une clé à l'objet `widget` pour extraire ce dernier, vous pouvez affecter une chaîne de texte en tant que second paramètre de la méthode **Add**.

L'objet **Collection** Visual Basic stocke tous ses éléments comme type **Object**, donc vous pouvez ajouter un élément de tout type de données. Il n'y a aucun dispositif de protection contre les types de données inappropriés qui sont ajoutés. Pour éviter cette limitation, vous pouvez utiliser les collections *génériques* de l'espace de noms System.Collections.Generic. Pour plus d'informations, consultez Comment : créer une collection d'objets.

Créer et détruire un objet de collection

Le mot clé New (Visual Basic) dans la déclaration pour la variable `widgetColl` entraîne la création d'un objet **Collection** lorsque le contrôle passe à l'instruction de déclaration. Comme **Collection** est une classe et non un type valeur, vous devez en créer une instance et conserver une référence à cette instance dans une variable. Cette instance est un objet Collection Visual Basic.

Comme tout autre objet, un objet **Collection** est marqué pour une opération garbage collection (GC) lorsque la dernière variable qui contient une référence à cet objet a la valeur Nothing (Visual Basic) ou devient hors de portée. Toutes les références d'objet qu'il contient sont libérées lorsqu'il est récupéré par l'opération garbage collection. C'est la raison pour laquelle la variable `widgetColl` dans l'exemple précédent est déclarée dans la classe parente afin d'exister pendant toute la durée de vie du programme.

Une collection maintient des références aux objets qu'elle contrôle, mais ne contient pas les objets eux-mêmes. Par conséquent, supprimer un objet **Collection** ne supprime pas les objets qu'elle contrôle. Chaque objet individuel qui avait été un élément de la collection continue d'exister jusqu'à ce qu'il soit marqué individuellement pour une opération garbage collection.

Travailler avec des éléments

Les services de base (ajout, suppression et extraction d'éléments d'une collection) dépendent des clés et des index. Une *clé* est une valeur de type **String**. Cette valeur peut être un nom, un numéro de permis de conduire, un numéro de téléphone ou un entier converti en une chaîne. La méthode **Add** vous permet d'associer une clé à un élément, comme cela est décrit dans Comment : ajouter, supprimer et récupérer des éléments d'une collection.

Un *index* de la classe **Collection** est un entier compris entre 1 et le nombre d'éléments de la collection. Le Count, propriété (objet Collection) retourne le nombre actuel d'éléments. Vous pouvez contrôler la valeur initiale d'index d'un élément à l'aide des paramètres *Before* ou *After* lorsque vous appelez **Add** ; cette valeur peut toutefois changer lors de l'ajout ou de la suppression d'autres éléments. Pour plus d'informations, consultez Add, méthode (objet Collection).

Vous pouvez supprimer un seul élément d'une collection en passant sa clé ou son index à Remove, méthode (objet Collection). Vous pouvez vider une collection et supprimer tous les éléments à l'aide de Clear, méthode (objet Collection).

Accès aux éléments

Vous pouvez passer une valeur des touches à Contains, méthode (objet Collection) pour tester si une collection contient un élément avec cette clé. Vous pouvez récupérer un élément en passant sa clé ou son index à Item, propriété (objet Collection).

Vous pouvez utiliser des valeurs d'index et la propriété **Item** pour parcourir les éléments d'une collection ou vous pouvez utiliser For Each...Next, instruction (Visual Basic). L'exemple suivant illustre deux possibilités pour octroyer une augmentation de 10 % à tous les employés de la collection d'objets `employee`, si la variable `employeesColl` contient une référence à un objet **Collection**.

```
Option Strict On
' The following alternative uses the Count and Item properties.
Dim emp As employee
For counter As Integer = 1 To employeesColl.Count
    emp = CType(employeesColl.Item(counter), employee)
    emp.payRate *= 1.1
Next counter
' The following alternative uses the For Each...Next statements.
For Each emp As employee In employeesColl
    emp.payRate *= 1.1
Next emp
```

Toutefois, si vous avez ajouté un ou plusieurs éléments à `employeesColl` qui ne sont pas de type `employee`, la boucle **For Each** lève une exception ArgumentException au moment de l'exécution.

Type de données des éléments

Un objet Collection Visual Basic stocke chaque élément avec un type de données **Object**. Par conséquent, la plage de types de données que vous pouvez ajouter à un objet **Collection** est la même que la plage des types de données que vous pouvez stocker dans une variable **Object**. Vous pouvez par exemple ajouter des types de données, des objets et des tableaux standard, ainsi que des structures et des instances de classe définies par l'utilisateur.

Comme l'objet **Collection** stocke chaque élément comme un **Object**, la propriété **Item** retourne une valeur **Object**. Pour utiliser l'élément dans votre code, vous devez habituellement convertir de **Object** au type de données exécutable de l'élément. La méthode que vous employez dépend du paramètre du commutateur de vérification de type dans Option Strict, instruction.

Convertir implicitement d'un objet

Si **Option Strict** a la valeur **Off**, vous pouvez convertir implicitement un élément de **Collection** en son type de données approprié, comme le montre l'exemple suivant.

```
Option Strict Off
Dim sampleColl As New Microsoft.VisualBasic.Collection()
Dim sampleString As String = "This is a string"
Dim aString As String
sampleColl.Add(sampleString)
' The following statements convert the collection item to a string.
Try
    aString = sampleColl.Item(1)
Catch ex As Exception
    ' Insert code to run if the collection item cannot be converted to String.
End Try
```

Convertir explicitement d'un objet

Si **Option Strict** a la valeur **On**, vous devez convertir explicitement de **Object** au type de données exécutable de l'élément. Pour obtenir un élément de **Item** de cette manière, vous pouvez utiliser le CType, fonction pour effectuer la conversion, comme le montre l'exemple suivant.

```
Option Strict On
Dim sampleColl As New Microsoft.VisualBasic.Collection()
Dim sampleString As String = "This is a string"
Dim aString As String
sampleColl.Add(sampleString)
' The following statements convert the collection item to a string.
Try
    aString = CType(sampleColl.Item(1), String)
Catch ex As Exception
    ' Insert code to run if the collection item cannot be converted to String.
```

```
End Try
```

Services supplémentaires

Les propriétés et les méthodes de l'objet **Collection** ne fournissent que les services de base pour les collections. Par exemple, la méthode **Add** ne peut pas vérifier le type d'un élément ajouté à une collection, alors que vous pourriez vouloir procéder à cette vérification pour garantir que la collection ne comporte qu'un seul type d'élément. Si vous pouvez garantir cela dans la méthode **Add**, vous disposez d'une collection *fortement typée*, et il n'est pas nécessaire de convertir le retour de la propriété **Item** en son type de données exécutable. Les performances s'en trouvent améliorées.

Vous pouvez fournir des fonctionnalités plus fiables - et des propriétés, des méthodes et des événements supplémentaires - en créant votre propre classe de collection, comme cela est décrit dans Comment : définir des collections dans vos classes.

Voir aussi

Référence

Collection, objet (Visual Basic)

Concepts

Collections en tant qu'alternative aux tableaux

Collections en Visual Basic

Comment : définir des collections dans vos classes

Vous pouvez ajouter une collection à l'une de vos classes pour gérer des groupes d'objets utilisés par votre classe. Pour cela, la solution la plus simple consiste à ajouter une variable publique de type **Collection** à votre classe. Prenons l'exemple d'une classe hypothétique nommée `widgetRepository` qui gère et expose des objets `widget`. Vous pouvez créer une variable `widgetColl` pour faire référence à une collection `widget`, comme discuté dans la procédure suivante.

Définition d'une collection simple

Pour définir une collection simple dans une classe

- Créez une variable publique assumant la fonction de collection pour vos objets.

```
Public Class widgetRepository
    Public widgetColl As New Microsoft.VisualBasic.Collection()
    ' Insert code to implement additional functionality.
End Class
```

La classe `widgetRepository` possède désormais une collection publique à laquelle peuvent être ajoutés des objets `widget`. Vous pouvez utiliser ensuite une For Each...Next, instruction (Visual Basic) pour traiter les éléments de la collection, comme le montre le code suivant.

```
For Each aWidget As widget In widgetColl
    ' Insert code to process widgetColl elements
Next aWidget
```

La collection `widgetColl` définie dans l'exemple précédent n'est pas *fortement typée*, ce qui signifie que vous pouvez lui ajouter tout type d'objet, pas seulement des objets `widget`. Cela peut entraîner des problèmes de sécurité de type. Par exemple, supposez que vous ajoutez une **String** à la collection, comme dans le code suivant.

```
Dim notWidget As String = "This is not a widget object!"
widgetColl.Add(notWidget)
```

Si vous faites cela, la boucle **For Each** de la procédure précédente lève une exception ArgumentException au moment de l'exécution car un élément de la collection n'est pas de type `widget`.

Nous vous conseillons de définir une classe générique lorsque vous souhaitez garantir la sécurité de type. Pour obtenir des informations supplémentaires et un exemple, consultez Comment : définir des collections de type sécurisé.

Voir aussi

Tâches

Comment : définir des collections de type sécurisé

Référence

Option Strict, instruction

Concepts

Classe de la collection Visual Basic

Collections en Visual Basic

Types génériques en Visual Basic

Comment : définir des collections de type sécurisé

Vous pouvez définir et créer une collection à l'aide de la classe **Collection** fournie par Visual Basic, comme l'exemple suivant l'illustre.

VB

```
Public Class widgetRepository
    Public widgetColl As New Microsoft.VisualBasic.Collection()
    ' Insert code to implement additional functionality.
End Class
```

Toutefois, cette collection `widgetColl` n'est pas *fortement typée*. Vous pouvez lui ajouter n'importe quel type d'élément, pas seulement des objets `widget`. Lorsque vous récupérez un élément, vous devrez peut-être le convertir en `widget`. Cela peut entraîner des problèmes de *sécurité de type*. Par exemple, supposez que vous ajoutez une **String** à la collection, à l'aide du code suivant.

VB

```
Dim notWidget As String = "This is not a widget object!"
widgetColl.Add(notWidget)
```

Dans ce cas, toute tentative ultérieure pour récupérer cet élément lève une exception ArgumentException au moment de l'exécution car l'élément de la collection n'est pas de type `widget`.

Protection contre les problèmes de sécurité de type

Nous vous recommandons d'optimiser la sécurité de type en définissant une *classe générique*. Elle permet d'appliquer un typage fort et de fournir une flexibilité dans le type de données particulier qu'elle utilise. Pour plus d'informations, consultez Types génériques en Visual Basic.

Pour définir une collection de type sécurisé dans une classe

- Utilisez l'une des classes génériques contenues dans l'espace de noms System.Collections.Generic, par exemple List, pour votre classe de collection. Vous pouvez ensuite créer une collection qui est restreinte uniquement aux membres `widget`. L'exemple suivant explique comment la déclaration dans l'exemple précédent peut être modifiée pour créer une collection générique.

  VB

  ```
  Public widgetColl As New System.Collections.Generic.List(Of widget)
  ```

  Dans ce cas, le typage fort permet d'ajouter uniquement des éléments `widget` à la collection, et chaque élément récupéré par l'intermédiaire de la propriété `Item` est un objet `widget`. Grâce à ce typage fort, le code de récupération peut utiliser toutes les propriétés et les méthodes exposées par `widget`.

  - ou -

- Créez votre propre classe de collection sans utiliser de classes prédéfinies. Appliquez une restriction à votre méthode **Add** pour accepter uniquement des objets `widget`, et implémentez votre propriété **Item** avec le type de retour `widget`. Pour plus d'informations, consultez Comment : définir des collections de type sécurisé.

  Si **Item** a retourné des éléments de type **Object**, vous disposez uniquement d'un accès explicite aux propriétés et aux méthodes définies sur la classe **Object**. Pour accéder aux membres `widget`, vous devrez activer **Option Strict Off** ou utiliser CType, fonction pour convertir explicitement l'élément retourné en `widget`, comme dans le code suivant.

  VB

  ```
  Dim nextWidget As widget
  Try
  ```

```
    nextWidget = CType(widgetColl.Item(1), widget)
Catch ex As Exception
    ' Insert code to run if the collection item is not a widget.
End Try
```

Dans l'un ou l'autre des cas, l'utilisation d'éléments **Object** oblige Visual Basic à utiliser une *liaison tardive*, ce qui altère vos performances.

Notez que, étant donné que la classe **Collection** de Visual Basic accepte et retourne des éléments **Object**, elle présente également les inconvénients du typage faible et de la liaison tardive.

Voir aussi

Référence

Option Strict, instruction
System.Collections
System.Collections.Generic
System.Collections.Specialized

Concepts

Classe de la collection Visual Basic
Collections en Visual Basic
Types génériques en Visual Basic

Comment : créer une collection d'objets

Comme avec tout objet, vous déclarez une variable pour contenir l'objet, puis vous créez l'objet de collection et l'assignez à la variable.

Pour un objet de collection, vous pouvez utiliser la classe de collection Visual Basic ou une classe de collection du .NET Framework. En particulier, vous pouvez créer une collection *générique* en utilisant une des classes de l'espace de noms System.Collections.Generic. Une collection générique est utile lorsque chaque élément de la collection a le même type de données. Les collections génériques mettent en vigueur le *typage fort* en autorisant uniquement l'ajout du type de données souhaité. Pour plus d'informations, consultez Comment : définir des collections de type sécurisé.

Une fois l'objet de collection créé, vous pouvez ajouter et supprimer des éléments et accéder aux éléments de la collection.

Deux exemples de création de collections suivent. Chaque collection contient des éléments **String** et associe une clé **String** à chaque élément. Les deux premières procédures créent une collection à l'aide de la classe de collection Visual Basic. Les deux dernières procédures créent une collection à l'aide d'une classe de collection générique du .NET Framework.

Pour créer une collection à l'aide de la classe de collection Visual Basic

1. Déclarez et créez une variable **Collection** Visual Basic, comme le montre l'exemple suivant.

   ```
   Dim sampleVisualBasicColl As New Microsoft.VisualBasic.Collection()
   ```

   La collection dans `sampleVisualBasicColl` peut accepter des éléments de tout type de données.

2. Utilisez la Add, méthode (objet Collection) pour ajouter des éléments à la collection. L'exemple suivant crée quatre éléments **String** et les ajoute à la collection. Il crée une valeur **String** unique comme clé pour chaque nouvel élément et passe cette valeur à la méthode **Add**.

   ```
   Dim item1, item2, item3, item4 As String
   item1 = "Items"
   item2 = "In"
   item3 = "A"
   item4 = "Collection"
   sampleVisualBasicColl.Add(item1, "firstkey")
   sampleVisualBasicColl.Add(item2, "secondkey")
   sampleVisualBasicColl.Add(item3, "thirdkey")
   sampleVisualBasicColl.Add(item4, "fourthkey")
   ```

   L'argument *Key* est facultatif dans une collection Visual Basic.

3. Si vous souhaitez supprimer un élément de la collection, vous pouvez utiliser la Remove, méthode (objet Collection), en identifiant l'élément par son index de position ou par sa clé facultative. L'exemple suivant illustre ce comportement :

   ```
   ' Remove the first element of the Visual Basic collection.
   sampleVisualBasicColl.Remove(1)
   ' Remove the element with the key "secondkey".
   sampleVisualBasicColl.Remove("secondkey")
   ```

   Notez que lorsqu'un élément est supprimé d'une **Collection** Visual Basic, les valeurs d'index sont renumérotées de 1 à la valeur de la Count, propriété (objet Collection).

Pour utiliser l'instruction For Each...Next afin de traiter les éléments de votre collection Visual Basic

1. Déclarez une variable du type stocké dans la collection. Pour l'exemple précédent, déclarez une variable de type **String**, comme le montre l'exemple suivant.

   ```
   ' Insert code from the preceding example.
   Dim aString As String
   ```

2. Utilisez une instruction For Each...Next (Visual Basic) pour examiner chaque élément de la collection. L'exemple suivant recherche une chaîne particulière et l'affiche s'il la trouve.

```
For Each aString in sampleVisualBasicColl
    If aString = "Collection" Then
        MsgBox(aString)
    End If
Next aString
```

Pour créer une collection à l'aide d'une classe de collection générique

1. Déclarez et créez une variable System.Collections.Generic.Dictionary du .NET Framework, comme le montre l'exemple suivant.

```
Dim sampleGenericColl As New System.Collections.Generic.Dictionary(Of String, String)
```

La variable `sampleGenericColl` contient une collection *de type sécurisé* qui accepte uniquement des éléments et des clés de type **String**.

2. Utilisez la méthode System.Collections.Generic.Dictionary.Add(pour ajouter des éléments à la collection. L'exemple suivant crée quatre éléments **String** et les ajoute à la collection. Il crée une valeur **String** unique comme clé pour chaque nouvel élément et passe cette valeur à la méthode **Add**.

```
Dim item1, item2, item3, item4 As String
item1 = "Items"
item2 = "In"
item3 = "A"
item4 = "Collection"
sampleGenericColl.Add("firstkey", item1)
sampleGenericColl.Add("secondkey", item2)
sampleGenericColl.Add("thirdkey", item3)
sampleGenericColl.Add("fourthkey", item4)
```

L'argument *Key* est requis dans cette collection générique.

3. Pour supprimer un élément de la collection, utilisez la méthode System.Collections.Generic.IDictionary.Remove(. Vous devez fournir la clé pour identifier l'élément à supprimer. L'exemple suivant illustre ce comportement :

```
If Not sampleGenericColl.Remove("thirdkey")
    ' Insert code to handle "thirdkey" not found in collection.
End If
```

Vous pouvez utiliser une instruction **For Each...Next** pour parcourir en boucle et traiter les éléments d'une collection, comme le démontre la procédure suivante.

Pour utiliser l'instruction For Each...Next afin de traiter les éléments de votre collection générique

1. Déclarez une variable du type stocké dans la collection. Pour l'exemple précédent, déclarez une variable de type **String**, comme le montre l'exemple suivant.

```
' Insert code from the preceding example.
Dim aPair As KeyValuePair(Of String, String)
```

2. Utilisez une instruction For Each...Next (Visual Basic) pour examiner chaque élément de la collection. L'exemple suivant recherche une chaîne particulière et l'affiche s'il la trouve.

```
For Each aPair In sampleGenericColl
    If aPair.Value = "Items" Then
        MsgBox(aPair.Key & " -- " & aPair.Value)
    End If
Next aPair
```

Voir aussi

Tâches

Comment : créer un tableau d'objets

Référence

Collection, objet (Visual Basic)
System.Collections
System.Collections.Generic
System.Collections.Specialized

Concepts

Classe de la collection Visual Basic

Comment : créer un tableau d'objets

Chaque objet est un type référence. La déclaration et l'utilisation d'un tableau d'un type référence sont identiques à celles d'un tableau d'un type de données quelconque. Les éléments d'un tableau de type d'objet peuvent être extraits par leur index et peuvent être manipulés comme tout objet de ce type.

Les tableaux peuvent posséder des fonctionnalités intégrées pour effectuer des recherches et des tris accessibles via la variable tableau. Pour plus d'informations sur ces méthodes, consultez Array.

Pour créer un tableau d'objets

1. Déclarez le tableau comme indiqué dans l'exemple suivant. Dans la mesure où les tableaux sont de base zéro, ils contiennent un élément supplémentaire par rapport à la limite supérieure déclarée.

```
Dim x(10) As widget
' x now contains 11 elements of type widget, x(0) through x(10).
```

2. Créez chaque élément du tableau ou assignez à chacun des éléments une référence à un objet déjà existant. C'est ce qu'illustre l'exemple suivant.

```
' Create each element of an array by using a loop.
For q As Integer = 0 To 10
    x(q) = New widget()
Next q
' Assign a reference to an existing object to two array elements.
Dim specialWidget As New widget()
x(0) = specialWidget
x(1) = specialWidget
```

   Notez que vous pouvez assigner des références au même objet à des différents éléments du tableau.

Voir aussi

Tâches

Comment : créer une collection d'objets

Concepts

Gestion de groupes d'objets

Types valeur et types référence

Autres ressources

Tableaux dans Visual Basic

Comment : ajouter, supprimer et récupérer des éléments d'une collection

La classe **Collection** de Visual Basic comprend des fonctionnalités intégrées qui vous permettent d'ajouter, de supprimer et d'extraire des éléments.

- Vous pouvez ajouter un élément à une collection dès la création ou l'obtention de celui-ci, par exemple un nouveau client.
- Vous pouvez supprimer un élément d'une collection lorsqu'il n'appartient plus à celle-ci, par exemple lorsqu'un employé quitte votre société.
- Vous pouvez extraire un élément d'une collection pour modifier son contenu, par exemple pour modifier le numéro de téléphone d'un stagiaire.

Remarque

Les objets **Collection** mettent à jour automatiquement leurs numéros d'index numériques lorsque vous ajoutez et supprimez des éléments. De ce fait, l'index numérique d'un élément donné peut être modifié souvent. Par conséquent, n'enregistrez pas une valeur d'index numérique qui procédera ultérieurement à l'extraction du même élément dans votre programme. Vous devez utiliser des clés à cette fin.

Pour ajouter un élément à une collection

- Utilisez le Add, méthode (objet Collection) et spécifiez l'élément par sa *Key*.

  ```
  object.Add(Item, Key [, {Before | After}])
  ```

  Par exemple, pour ajouter un objet ordre de travail (work order) à une collection d'ordres de travail en utilisant la propriété `ID` de l'ordre de travail en tant que clé, vous pouvez effectuer l'appel suivant.

  ```
  workOrders.Add(woNew, woNew.ID)
  ```

  L'appel précédent suppose que la propriété `ID` est une chaîne. S'il s'agit d'un nombre (par exemple, un entier de type **Long**), utilisez sa méthode ToString pour le convertir en une valeur de type **String** requise pour l'argument *Key*.

  ```
  workOrders.Add(woNew, woNew.ID.ToString())
  ```

  L'utilisation d'une clé est facultative. Si vous ne souhaitez pas associer une clé à l'objet dans votre collection, vous pouvez l'ajouter sans clé.

  ```
  workOrders.Add(woNew)
  ```

  Vous pouvez utiliser les arguments *Before* et *After* facultatifs pour gérer une collection ordonnée d'objets. L'élément à ajouter est placé dans la collection avant ou après l'élément identifié par l'argument *Before* ou *After*, respectivement. Par exemple, le fait d'attribuer la valeur 1 à *Before* insère un élément au début de la collection puisque les objets **Collection** sont de base 1.

  ```
  workOrders.Add(woNew, woNew.ID, 1)
  ```

  De même, l'argument *After* ajoute un élément après l'index spécifié. L'exemple suivant ajoute un élément en tant que troisième élément.

  ```
  workOrders.Add(woNew, woNew.ID,,2)
  ```

Vous pouvez spécifier une valeur soit pour *Before*, soit pour *After*, mais pas pour ces deux arguments.

Pour supprimer un élément d'une collection

- Utilisez Remove, méthode (objet Collection) et spécifiez l'élément soit par son *Index*, soit par sa *Key*.

  ```
  object.Remove({Index | Key})
  ```

  L'argument *Index* est la position de l'élément que vous souhaitez supprimer. L'argument *Key* est la même chaîne que vous avez utilisée pour ajouter l'élément à la collection. Si la clé du troisième élément d'une collection est "W017493", vous pouvez utiliser l'une des deux instructions suivantes pour la supprimer.

  ```
  workOrders.Remove(3)
  workOrders.Remove("W017493")
  ```

Pour supprimer tous les éléments d'une collection

- Utilisez Clear, méthode (objet Collection).

  ```
  object.Clear()
  ```

  La méthode **Clear** vide la collection.

Pour extraire un élément d'une collection

1. Utilisez Item, propriété (objet Collection) et spécifiez l'élément soit par son *Index*, soit par sa *Key*.

   ```
   variable = object.Item({Index | Key})
   ```

   Comme avec la méthode **Remove**, l'argument *Index* est la position de l'élément dans la collection, et l'argument *Key* est la chaîne utilisée lorsque l'élément a été ajouté. En utilisant le même exemple que pour la méthode **Remove**, vous pouvez utiliser l'une des deux instructions suivantes pour extraire le troisième élément de la collection.

   ```
   woCurrent = workOrders.Item(3)
   woCurrent = workOrders.Item("W017493")
   ```

   Remarque

   Si vous utilisez des nombres comme clés, vous devez utiliser leurs méthodes ToString pour les convertir en chaînes avant de les passer à la méthode **Add** ou **Remove**, ou à la propriété **Item**. Un objet **Collection** de Visual Basic suppose toujours qu'un nombre est un index plutôt qu'une chaîne clé.

2. Si vous connaissez la clé au moment de la compilation, vous pouvez utiliser également l'opérateur d'accès de type dictionnaire (**!**) pour accéder à un élément de la collection sans placer la clé entre guillemets ou parenthèses. L'appel précédent peut s'écrire de la manière suivante.

   ```
   woCurrent = workOrders!W017493
   ```

Voir aussi

Référence

Collection, membres de l'objet
Add, méthode (objet Collection)
Remove, méthode (objet Collection)
Clear, méthode (objet Collection)
Item, propriété (objet Collection)

Concepts

Classe de la collection Visual Basic
Collections en tant qu'alternative aux tableaux
Collections en Visual Basic

Comment : parcourir une collection dans Visual Basic

Vous pouvez utiliser une boucle **For Each** pour parcourir chaque élément d'une collection.

Exemple

L'exemple suivant utilise l'For Each...Next, instruction (Visual Basic) pour accéder à chaque élément d'une collection.

```
Dim testCollection As New Microsoft.VisualBasic.Collection()
' The collection is empty until you add one or more items to it.
For Each collectionItem As Object In testCollection
    ' Perform desired processing on each item.
Next collectionItem
```

Compilation du code

Cet exemple nécessite :

- un accès à l'espace de noms System.

Voir aussi

Tâches

Comment : ajouter, supprimer et récupérer des éléments d'une collection

Référence

For Each...Next, instruction (Visual Basic)

Concepts

Collections en Visual Basic

Dépannage des collections

Cette page répertorie quelques problèmes courants qui peuvent se produire lors de l'utilisation des collections.

Utilisation d'un type de collection incorrect

Plusieurs types de collection sont disponibles pour un développeur Visual Basic : la classe **Collection** Visual Basic et les classes de collection fournies par le .NET Framework. Ces classes ne sont pas compatibles entre elles. En d'autres termes, si vous déclarez une variable comme étant d'un type de collection, vous ne pouvez pas lui assigner un objet d'un autre type. De même, vous pouvez accéder uniquement aux méthodes et aux propriétés du type de collection déclaré.

Les principales différences entre les classes de collection Visual Basic et celles du .NET Framework sont les suivantes :

- **Base d'Index.** Les collections du .NET Framework sont de *base zéro*, tandis que la collection Visual Basic est de *base 1*. Cela signifie que les éléments d'une collection Visual Basic ont des valeurs d'index comprises entre 1 et la valeur de Count, propriété (objet Collection), tandis que les éléments d'une collection du .NET Framework ont des valeurs d'index comprises entre 0 et une valeur inférieure d'une unité à la valeur de la propriété `Count` de la collection.
- **Type d'élément.** La collection Visual Basic prend en charge des éléments de type **Object**, qui n'est pas un *type sécurisé* car vous pouvez ajouter un élément à un type de données. Cela entraîne généralement une diminution des performances car le compilateur doit soumettre les éléments à une conversion *box* et *unbox* pour les convertir vers et à partir du type de données Object. Certaines collections du .NET Framework contiennent également des éléments de type **Object**, mais beaucoup d'autres sont *fortement typées*, c'est-à-dire qu'elles prennent en charge des éléments d'un type spécifique, ce qui leur confère un type sécurisé et optimise les performances.
- **Éléments de clé.** La collection Visual Basic vous permet de spécifier une *clé* lorsque vous lui ajoutez un élément. La clé est une valeur **String** unique que vous pouvez utiliser ultérieurement pour accéder à cet élément particulier. Les collections du .NET Framework varient en ce qui concerne les clés. Certaines prennent en charge des clés et d'autres pas.

Les espaces de noms qui contiennent les diverses définitions de classe de collection sont les suivantes :

- Microsoft.VisualBasic — la classe **Collection** Visual Basic
- System.Collections — les classes de collection spécifiques telles que les listes, les files d'attente, les tableaux de bits, les tables de hachage et les dictionnaires
- System.Collections.Generic — les classes de collection génériques qui vous permettent de créer des collections fortement typées et de spécifier le type de données d'élément lors de leur création
- System.Collections.Specialized — les classes de collection spécialisées et fortement typées, telles que les dictionnaires de liste liée et hybrides, les collections à vecteur de bits et d'objets de nom et les collections à chaîne uniquement

Approche correcte

Choisissez le type de collection qui convient le mieux à vos besoins. Déclarez votre variable de collection comme étant de ce type, et pensez à créer un objet de ce même type. Utilisez la qualification complète pour spécifier le type de collection prévu. L'exemple suivant affiche deux déclarations présentant une qualification complète.

VB

```
Dim customers As New Microsoft.VisualBasic.Collection()
Dim stringQueue As New System.Collections.Generic.Queue(Of String)
```

Une fois que vous avez créé une collection d'un type spécifique, utilisez uniquement les méthodes et les propriétés définies sur ce type. Définissez **Option Strict On** pour intercepter les assignations d'objet ou les accès de membre incorrects au moment de la compilation.

Voir aussi

Référence

Option Strict, instruction

Concepts

Collections en Visual Basic

Classe de la collection Visual Basic

Événements en Visual Basic

Un événement est un message envoyé par un objet qui annonce que quelque chose s'est produit. Les événements sont implémentés à l'aide de délégués, qui sont une forme de pointeur fonction orienté objet permettant à une fonction d'être appelée indirectement par l'intermédiaire d'une référence à la fonction.

Dans cette section

Événements et gestionnaires d'événements

Propose une vue d'ensemble des événements, des gestionnaires d'événements et de la terminologie associée.

WithEvents et la clause Handles

Indique comment utiliser le mot clé **WithEvents** et la clause **Handles** pour associer un événement à un gestionnaire d'événements.

AddHandler et RemoveHandler

Indique comment utiliser les instructions **AddHandler** et **RemoveHandler** pour associer de manière dynamique un événement à un gestionnaire d'événements.

Comment : ajouter des événements à une classe

Décrit le processus de définition des événements.

Comment : créer des gestionnaires d'événements dans l'éditeur de code Visual Basic

Explique comment utiliser l'éditeur de code de Visual Basic pour écrire du code de réponse aux événements.

Comment : déclencher un événement (Visual Basic)

Montre comment définir un événement et utilise l'instruction **RaiseEvent** pour le déclencher.

Comment : créer un événement et un gestionnaire (Visual Basic)

Montre comment définir un événement et un gestionnaire d'événements, et comment utiliser l'instruction **AddHandler** pour les associer.

Comment : écrire des gestionnaires d'événements

Montre comment écrire des gestionnaires d'événements à l'aide de la clause **Handles** ou de l'instruction **AddHandler**.

Comment : gérer les événements dans Visual Basic

Définit une classe qui déclenche un événement lorsque vous appelez la méthode **CauseEvent**.

Procédure pas à pas : déclaration et déclenchement des événements

Vous guide pas à pas dans le processus de déclaration et de déclenchement des événements pour une classe.

Procédure pas à pas : gestion des événements

Montre comment écrire une procédure gestionnaire d'événements.

Comment : déclarer des événements qui évitent les blocages

Montre comment définir un événement personnalisé qui autorise l'appel asynchrone de ses gestionnaires d'événements.

Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Montre comment définir un événement personnalisé qui utilise la mémoire uniquement lorsque l'événement est géré.

Dépannage des gestionnaires d'événements hérités en Visual Basic .2005

Répertorie les problèmes courants liés aux gestionnaires d'événements dans les composants hérités.

Rubriques connexes

Délégués en Visual Basic

Vue d'ensemble de délégués en Visual Basic.

Gestion et déclenchement d'événements

Propose une vue d'ensemble du modèle d'événement dans le .NET Framework.

Création de gestionnaires d'événements dans les Windows Forms

Décrit comment utiliser des événements associés aux objets Windows Forms.

Événements et gestionnaires d'événements

Vous pouvez comparer un projet Visual Studio à une série de procédures qui s'exécutent dans une séquence, mais en réalité, la plupart des programmes sont pilotés par des événements, c'est-à-dire que le déroulement de l'exécution dépend d'occurrences externes appelées *événements*.

Un événement est un signal qui informe l'application que quelque chose d'important s'est produit. Par exemple, lorsqu'un utilisateur clique sur un contrôle d'un formulaire, le formulaire peut déclencher un événement **Click** et appeler une procédure qui gère l'événement. Les événements séparent également les tâches pour communiquer. Supposons, par exemple, que votre application exécute une tâche de tri séparément de l'application principale. Si un utilisateur annule le tri, votre application peut envoyer un événement d'annulation demandant au processus de tri de s'interrompre.

Termes et concepts relatifs aux événements

Cette section décrit les termes et concepts utilisés avec les événements en Visual Basic.

Déclaration d'événements

Vous déclarez des événements dans le cadre de classes, structures, modules et interfaces à l'aide du mot clé **Event**, comme dans l'exemple suivant :

VB

```
Event AnEvent(ByVal EventNumber As Integer)
```

Déclenchement d'événements

Un événement est comparable à un message annonçant que quelque chose d'important s'est produit. L'acte de diffusion du message est appelé *déclenchement* de l'événement. En Visual Basic, vous déclenchez des événements à l'aide de l'instruction **RaiseEvent**, comme dans l'exemple suivant :

VB

```
RaiseEvent AnEvent(EventNumber)
```

Les événements doivent être déclenchés dans la portée de la classe, du module ou de la structure où ils sont déclarés. Par exemple, une classe dérivée ne peut pas déclencher d'événements hérités à partir d'une classe de base.

Émetteurs d'événements

Tout objet capable de déclencher un événement est un *émetteur d'événements*, également appelé *source d'événements*. Les formulaires, contrôles et objets définis par l'utilisateur sont des exemples d'émetteurs d'événements.

Gestionnaires d'événements

Les *gestionnaires d'événements* sont des procédures appelées lorsqu'un événement correspondant se produit. Vous pouvez utiliser n'importe quelle sous-routine valide avec une signature correspondante comme gestionnaire d'événements. Il vous est cependant impossible d'utiliser une fonction en tant que gestionnaire d'événements parce qu'une fonction ne peut retourner une valeur à la source des événements.

Visual Basic utilise pour les gestionnaires d'événements une convention d'affectation de noms standard qui associe le nom de l'émetteur de l'événement, un trait de soulignement et le nom de l'événement. Par exemple, l'événement **Click** d'un bouton appelé `button1` aurait pour nom `Sub button1_Click`.

Remarque

Nous vous recommandons, mais ce n'est pas obligatoire, d'utiliser cette convention d'affectation de noms lorsque vous défin issez des gestionnaires d'événements pour vos propres événements ; vous pouvez utiliser n'importe quel nom de sous-routi ne valide.

Association d'événements aux gestionnaires d'événements

Avant de pouvoir utiliser un gestionnaire d'événements, vous devez au préalable l'associer à un événement à l'aide de l'instruction **Handles** ou **AddHandler**.

L'instruction **WithEvents** et la clause **Handles** assurent une méthode déclarative de spécification des gestionnaires d'événements. Les événements déclenchés par un objet déclaré par **WithEvents** peuvent être gérés par n'importe quelle sous-routine avec une clause **Handles** qui nomme ces événements. Même si la clause **Handles** constitue la méthode standard d'association d'un événement à un gestionnaire d'événements, son action est restreinte à l'association d'événements aux gestionnaires d'événements au moment de la compilation.

Les instructions **AddHandler** et **RemoveHandler** sont plus souples à utiliser que la clause **Handles**. Elles vous permettent de connecter et déconnecter dynamiquement les événements par rapport à au moins un gestionnaire d'événements au moment de l'exécution et ne vous obligent pas à déclarer des variables d'objets à l'aide de **WithEvents**. Toutefois, des restrictions s'appliquent à l'utilisation de **WithEvents**. Pour plus d'informations, consultez WithEvents et la clause Handles.

Dans certains cas, par exemple dans celui d'événements associés à des formulaires ou contrôles, Visual Basic choisit automatiquement un gestionnaire d'événements vide et l'associe à un événement. Par exemple, lorsque vous double-cliquez sur un bouton de commande dans un formulaire en mode design, Visual Basic crée un gestionnaire d'événements vide et une variable **WithEvents** pour le bouton de commande, comme dans le code suivant :

VB

```
Friend WithEvents Button1 As System.Windows.Forms.Button
Protected Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

Voir aussi

Tâches

Comment : écrire des gestionnaires d'événements

Référence

Handles

AddHandler, instruction

Concepts

AddHandler et RemoveHandler

WithEvents et la clause Handles

WithEvents et la clause Handles

L'instruction **WithEvents** et la clause **Handles** assurent une méthode déclarative de spécification des gestionnaires d'événements. Un événement déclenché par un objet déclaré par le mot clé **WithEvents** peut être géré par n'importe quelle procédure avec une instruction **Handles** pour cet événement, comme le montre l'exemple suivant :

VB

```
' Declare a WithEvents variable.
Dim WithEvents EClass As New EventClass

' Call the method that raises the object's events.
Sub TestEvents()
    EClass.RaiseEvents()
End Sub

' Declare an event handler that handles multiple events.
Sub EClass_EventHandler() Handles EClass.XEvent, EClass.YEvent
    MsgBox("Received Event.")
End Sub

Class EventClass
    Public Event XEvent()
    Public Event YEvent()
    ' RaiseEvents raises both events.
    Sub RaiseEvents()
        RaiseEvent XEvent()
        RaiseEvent YEvent()
    End Sub
End Class
```

L'instruction **WithEvents** et la clause **Handles** représentent souvent la meilleure solution pour les gestionnaires d'événements, parce que la syntaxe déclarative qu'ils utilisent simplifie beaucoup le codage, la lecture et le débogage de la gestion des événements. Il faut cependant garder à l'esprit les restrictions liées à l'utilisation des variables **WithEvents** :

- Vous ne pouvez pas utiliser une variable **WithEvents** comme variable d'objet. Cela signifie que vous ne pouvez pas la déclarer en tant que **Object** ; vous devez spécifier le nom de la classe lorsque vous déclarez la variable.
- Les événements partagés étant associés à des instances de classe, vous ne pouvez pas utiliser **WithEvents** pour les gérer de façon déclarative. De même, vous ne pouvez pas utiliser **WithEvents** ou **Handles** pour gérer les événements de **Structure**. Dans les deux cas, vous pouvez utiliser l'instruction **AddHandler** pour gérer ces événements.
- Vous ne pouvez pas créer de tableaux de variables **WithEvents**.
- Les variables **WithEvents** autorisent un seul gestionnaire d'événements à gérer un ou plusieurs types d'événements ou un ou plusieurs gestionnaires d'événements à gérer le même type d'événements.

Voir aussi

Référence

Handles
WithEvents
AddHandler, instruction

Concepts

AddHandler et RemoveHandler

AddHandler et RemoveHandler

L'instruction **AddHandler** est comparable à la clause **Handles** dans la mesure où toutes deux vous permettent de spécifier un gestionnaire d'événements. Cependant, **AddHandler** utilisé conjointement avec **RemoveHandler**, offre une plus grande souplesse que la clause **Handles** ; ils vous permettent d'ajouter, de supprimer et de changer dynamiquement le gestionnaire d'événements associé à un événement. Si vous souhaitez gérer des événements partagés ou des événements d'une structure, vous devez utiliser **AddHandler**.

AddHandler prend en compte deux arguments : le nom d'un événement issu d'un émetteur d'événements tel qu'un contrôle et une expression qui a pour valeur un délégué. Comme l'instruction **AddressOf** retourne toujours une référence au délégué, vous n'êtes pas obligé de spécifier explicitement la classe déléguée lorsque vous utilisez **AddHandler**. L'exemple ci-dessous associe un gestionnaire d'événements à un événement déclenché par un objet :

VB

```
AddHandler Obj.XEvent, AddressOf Me.XEventHandler
```

RemoveHandler, qui déconnecte un événement d'un gestionnaire d'événements, utilise la même syntaxe que **AddHandler**. Par exemple :

VB

```
RemoveHandler Obj.XEvent, AddressOf Me.XEventHandler
```

Voir aussi

Tâches

Comment : écrire des gestionnaires d'événements

Référence

AddHandler, instruction

Concepts

Événements et gestionnaires d'événements

WithEvents et la clause Handles

Comment : ajouter des événements à une classe

Vous ajoutez des événements à une classe en les déclarant avec l'instruction **Events**. La déclaration comprend le nom de l'événement et les arguments qu'il utilise.

L'ajout d'un événement à une classe permet de spécifier qu'un objet de cette classe peut déclencher un événement spécifique. Pour provoquer l'occurrence effective d'un événement, vous devez utiliser l'instruction **RaiseEvent**. Vous pouvez utiliser le mot clé **Handles** ou l'instruction **AddHandler** pour associer l'événement à une procédure de gestionnaire d'événements. Les événements doivent être déclenchés dans la portée dans laquelle ils sont déclarés. Par exemple, une classe dérivée ne peut pas déclencher d'événements hérités à partir d'une classe de base.

Remarque
Les événements ne peuvent avoir de valeurs de retour, d'arguments facultatifs ou d'arguments **ParamArray**.

Pour ajouter un événement à une classe

- Dans la section des déclarations du module de classe qui définit la classe, utilisez l'instruction **Event** pour déclarer l'événement avec tous les arguments que vous souhaitez qu'il comporte. Par exemple :

  VB

```
Public Event PercentDone(ByVal Percent As Single, _
                         ByRef Cancel As Boolean)
```

Voir aussi

Tâches

Procédure pas à pas : déclaration et déclenchement des événements
Procédure pas à pas : gestion des événements
Comment : écrire des gestionnaires d'événements

Référence

RaiseEvent, instruction
Handles
AddHandler, instruction

Concepts

Événements et gestionnaires d'événements
Délégués et opérateur AddressOf
AddHandler et RemoveHandler

Autres ressources

Polymorphisme

Comment : créer des gestionnaires d'événements dans l'éditeur de code Visual Basic

L'**éditeur de code** Visual Basic simplifie la création de gestionnaires d'événements pour vos Windows Forms. Bien qu'il ne vous permette pas de lier plusieurs événements au même gestionnaire (consultez Comment : connecter plusieurs événements à un même gestionnaire d'événements dans les Windows Forms), il vous offre néanmoins une manière simple et rapide de créer des gestionnaires d'événements pendant que vous utilisez le code de votre formulaire, parce qu'il vous évite d'avoir à afficher le formulaire dans le Concepteur Windows Forms.

Pour créer un gestionnaire d'événements à l'aide de l'éditeur de code Visual Basic

1. Dans l'**éditeur de code**, dans la liste déroulante **Nom de la classe**, sélectionnez le formulaire ou le contrôle pour lequel vous voulez créer un gestionnaire d'événements.
2. Dans la liste **Nom de la méthode** située en haut de l'**éditeur de code**, sélectionnez l'événement pour lequel vous voulez créer un gestionnaire d'événements.

   Visual Basic crée un gestionnaire d'événements et l'ajoute à la classe de votre formulaire.

Voir aussi

Tâches

Comment : créer des gestionnaires d'événements à l'aide du concepteur
Dépannage des gestionnaires d'événements hérités en Visual Basic .2005

Concepts

Vue d'ensemble des gestionnaires d'événements (Windows Forms)

Autres ressources

Création de gestionnaires d'événements dans les Windows Forms
Création d'un nouveau Windows Form

Comment : déclencher un événement (Visual Basic)

Cet exemple définit un événement (`TimeExpired`) et utilise l'instruction **RaiseEvent** pour le déclencher.

Exemple

VB

```
Public Event TimeExpired(ByVal Status As String)
Public Sub RaiseTimeExpiredEvent()
    RaiseEvent TimeExpired("Your time has run out")
End Sub
```

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extraits de code, il se trouve dans **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

Compilation du code

Cet exemple nécessite les éléments suivants :

- l'accès aux membres de l'espace de noms System. Ajoutez une instruction **Imports** si vous n'utilisez pas des noms de membres qualifiés complets dans votre code. Pour plus d'informations, consultez Imports, instruction ;
- la présence de l'instruction **Event** au niveau de la classe et non pas à l'intérieur d'une procédure ;
- la présence de l'instruction **RaiseEvent** à l'intérieur d'une procédure dans votre application.

Voir aussi

Tâches

Comment : créer un événement et un gestionnaire (Visual Basic)

Référence

Event, instruction

RaiseEvent, instruction

Concepts

Événements et gestionnaires d'événements

Autres ressources

Événements en Visual Basic

Comment : créer un événement et un gestionnaire (Visual Basic)

Cet exemple définit un événement, `TimeExpired`, et un gestionnaire d'événements, `HandleTimeExpired`, et les associe à l'aide de l'instruction **AddHandler**.

Exemple

VB

```
Public Event TimeExpired(ByVal Status As String)
Public Sub HandleTimeExpired(ByVal Status As String)
    ' Perform desired processing for when time has expired.
    MsgBox("HandleTimeExpired caught the TimeExpired event" & _
           vbCrLf & "Status = " & Status)
End Sub
Public Sub SetUpEventHandler()
    AddHandler TimeExpired, AddressOf HandleTimeExpired
End Sub
```

Compilation du code

Cet exemple nécessite les éléments suivants :

- l'accès aux membres de l'espace de noms System. Ajoutez une instruction **Imports** si vous n'utilisez pas des noms de membres qualifiés complets dans votre code. Pour plus d'informations, consultez Imports, instruction ;
- la présence de l'instruction **Event** au niveau de la classe et non pas à l'intérieur d'une procédure ;
- la définition de l'instruction **Event** et des deux procédures (`HandleTimeExpired` et `SetUpEventHandler`) dans la même classe ou le même module. Sinon, l'instruction **AddHandler** doit qualifier l'événement et le gestionnaire d'événements vers les objets dans lesquels ils sont définis.

Voir aussi

Tâches

Comment : déclencher un événement (Visual Basic)

Référence

Event, instruction

AddHandler, instruction

Concepts

Événements et gestionnaires d'événements

Autres ressources

Événements en Visual Basic

Comment : écrire des gestionnaires d'événements

La façon dont vous construisez un gestionnaire d'événements dépend de celle dont vous voulez l'associer aux événements. La méthode standard de création d'un gestionnaire d'événements consiste à utiliser le mot clé **Handles** avec le mot clé **WithEvents**. Visual Basic offre une autre méthode de gestion des événements : l'instruction **AddHandler**. **AddHandler** et **RemoveHandler** vous permettent de démarrer et d'interrompre dynamiquement la gestion des événements pour un événement spécifique. Vous pouvez utiliser l'une ou l'autre de ces méthodes, mais pas à la fois **WithEvents** et **AddHandler** avec le même événement.

Gestion des événements à l'aide de WithEvents

Le mot clé **WithEvents** vous permet de créer des variables d'objets au niveau de la classe ou du module qui sont utilisables avec la clause **Handles** dans les gestionnaires d'événements.

Pour gérer les événements à l'aide de WithEvents et de la clause Handles

1. Créez une classe simple qui contient un événement.

   VB

   ```
   Class Class1
       Public Event AnEvent(ByVal EventNumber As Integer)
   End Class
   ```

2. Dans la classe ou le module destiné à gérer l'événement, utilisez le mot clé **WithEvents** pour déclarer une variable d'objet pour la source de vos événements, conformément à l'exemple suivant :

   VB

   ```
   Public WithEvents ClassInst As Class1
   ```

3. Dans l'éditeur de code, choisissez la variable **WithEvents** que vous venez de déclarer dans la liste déroulante **Nom de la classe** située à gauche.

4. Choisissez l'événement que vous souhaitez gérer dans la liste déroulante **Nom de la méthode** située à droite. L'éditeur de code crée la procédure gestionnaire d'événements vide avec une clause **Handles**.

   Remarque

   Cette étape est facultative. Vous pouvez créer manuellement la procédure gestionnaire d'événements à condition que l a procédure créée soit une sous-routine, contienne la liste d'arguments correspondant à l'événement géré et comporte une clause **Handles** spécifiant l'événement géré.

5. Ajoutez le code de gestion des événements à la procédure gestionnaire d'événements à l'aide des arguments fournis. Le code suivant en est un exemple :

   VB

   ```
   Public Sub ClassInst_AnEvent(ByVal EventNumber As Integer) _
         Handles ClassInst.AnEvent
       MsgBox("Received event number: " & CStr(EventNumber))
   End Sub
   ```

Gestion des événements à l'aide de AddHandler

Vous pouvez utiliser l'instruction **AddHandler** pour connecter dynamiquement les événements par rapport aux procédures gestionnaires d'événements.

Pour gérer des événements à l'aide de AddHandler

1. Créez une sous-routine pour gérer l'événement, comme dans l'exemple suivant :

   VB

   ```
   Public Sub EHandler(ByVal EventNumber As Integer)
       MsgBox("Received event number " & CStr(EventNumber))
   End Sub
   ```

2. Déclarez une variable d'objet de la classe qui est la source des événements que vous souhaitez gérer. Contrairement à une variable **WithEvents**, il peut s'agir d'une variable locale dans une procédure. Par exemple :

   VB

   ```
   Public Sub TestAddHandler()
       Dim CI As New Class1
   End Sub
   ```

3. Utilisez l'instruction **AddHandler** pour spécifier le nom de l'émetteur d'événements et l'instruction **AddressOf** pour fournir le nom de votre gestionnaire d'événements. Par exemple, ajoutez le code suivant à la fin de la sous-routine `TestAddHandler` :

   VB

   ```
   AddHandler CI.AnEvent, AddressOf EHandler
   ```

   N'importe quelle procédure peut servir de gestionnaire d'événements si elle prend en charge les bons arguments pour l'événement géré.

Utilisation de RemoveHandler pour arrêter la gestion des événements

Vous pouvez utiliser l'instruction **RemoveHandler** pour déconnecter dynamiquement les événements à partir des procédures gestionnaires d'événements.

Pour arrêter la gestion des événements en utilisant RemoveHandler

- Utilisez l'instruction **RemoveHandler** pour spécifier le nom de l'émetteur d'événements et l'instruction **AddressOf** pour fournir le nom de votre gestionnaire d'événements. La syntaxe des instructions **RemoveHandler** correspondra toujours étroitement à l'instruction **AddHandler** utilisée pour démarrer la gestion des événements. Par exemple :

  VB

  ```
  RemoveHandler CI.AnEvent, AddressOf EHandler
  ```

Gestion d'événements hérités d'une classe de base

Les *Classes dérivées*, classes qui héritent des caractéristiques d'une classe de base, peuvent gérer des événements déclenchés par leur classe de base à l'aide de l'instruction **Handles MyBase**.

Pour gérer les événements d'une classe de base

- Déclarez un gestionnaire d'événements dans la classe dérivée en ajoutant une instruction **Handles MyBase.***eventname* à la ligne de déclaration de votre procédure gestionnaire d'événements, où *eventname* est le nom de l'événement dans la classe de base gérée. Par exemple :

  VB

  ```
  Public Class BaseClass
  ```

```
    Public Event BaseEvent(ByVal i As Integer)
    ' Place methods and properties here.
End Class

Public Class DerivedClass
    Inherits BaseClass
    Sub EventHandler(ByVal x As Integer) Handles MyBase.BaseEvent
        ' Place code to handle events from BaseClass here.
    End Sub
End Class
```

Voir aussi

Référence

Handles

AddHandler, instruction

Concepts

Événements et gestionnaires d'événements

Délégués et opérateur AddressOf

AddHandler et RemoveHandler

Comment : gérer les événements dans Visual Basic

L'exemple suivant définit une classe qui déclenche un événement lorsque vous appelez la méthode `CauseEvent`. Cet événement est géré par une procédure gestionnaire d'événements appelée `EventHandler`.

Exemple

Pour exécuter cet exemple, ajoutez le code ci-dessous à la classe Form d'un projet d'application Windows Visual Basic et appelez la procédure `TestEvents` avec un argument de type nombre entier.

VB

```
Public Class Class1
    ' Declare an event for this class.
    Public Event Event1(ByVal EventNumber As Integer)
    ' Define a method that raises an event.
    Sub CauseEvent(ByVal EventNumber As Integer)
        RaiseEvent Event1(EventNumber)
    End Sub
End Class

Protected Sub TestEvents(ByVal EventNumber As Integer)
    Dim Obj As New Class1
    AddHandler Obj.Event1, AddressOf Me.EventHandler
    ' Cause the object to raise an event.
    Obj.CauseEvent(EventNumber)
End Sub

Sub EventHandler(ByVal EventNumber As Integer)
    MsgBox("Received event number " & EventNumber.ToString)
End Sub
```

Voir aussi

Tâches

Comment : écrire des gestionnaires d'événements

Concepts

Événements et gestionnaires d'événements

Délégués et opérateur AddressOf

AddHandler et RemoveHandler

Procédure pas à pas : déclaration et déclenchement des événements

Cette procédure pas à pas explique comment déclarer et déclencher des événements pour une classe nommée `Widget`. Après avoir réalisé les diverses étapes de la procédure, il peut s'avérer utile de lire la rubrique connexe, Procédure pas à pas : gestion des événements, qui montre comment utiliser des événements à partir d'objets `Widget` afin de fournir des informations d'état dans une application.

Classe Widget

Envisageons pour l'instant une classe du nom de `Widget`. Votre classe `Widget` comprend une méthode qui peut prendre du temps à s'exécuter, et vous voulez que votre application puisse afficher une sorte d'indicateur d'achèvement de cette tâche.

Il est clair que vous pourriez faire en sorte que l'objet `Widget` affiche une boîte de dialogue indiquant le pourcentage de réalisation de l'opération, mais cette boîte de dialogue apparaîtrait alors systématiquement dans tous les projets dans lesquels la classe `Widget` a été utilisée. Un principe utile à suivre en matière de design d'objets consiste à laisser l'application qui utilise un objet le soin de gérer l'interface utilisateur, à moins que l'objectif général de l'objet soit de gérer un formulaire ou une boîte de dialogue.

`Widget` a pour objectif d'effectuer d'autres tâches, c'est pourquoi il est préférable d'ajouter un événement `PercentDone` et de laisser à la procédure qui appelle les méthodes de la classe `Widget` le soin de gérer cet événement et d'afficher des mises à jour d'état. L'événement `PercentDone` peut également fournir un mécanisme d'annulation de la tâche.

Pour créer l'exemple de code de cette rubrique

1. Ouvrez un nouveau projet Application Windows Visual Basic, puis créez un formulaire nommé `Form1`.
2. Ajoutez deux boutons et une étiquette à `Form1`.
3. Nommez les objets de la façon indiquée dans le tableau suivant.

Objet	Propriété	Valeur
`Button1`	`Text`	Start Task
`Button2`	`Text`	Cancel
`Label`	`(Name)`, `Text`	lblPercentDone, 0

4. Dans le menu **Projet**, sélectionnez **Ajouter une classe** pour ajouter une classe nommée `Widget.vb` au projet.

Pour déclarer un événement pour la classe Widget

- Utilisez le mot clé **Event** pour déclarer un événement dans la classe `Widget`. Notez qu'un événement peut comporter des arguments **ByVal** et **ByRef**, comme l'illustre l'événement `PercentDone` de `Widget` :

  VB

```
Public Event PercentDone(ByVal Percent As Single, _
                         ByRef Cancel As Boolean)
```

Lorsque l'objet appelant reçoit un événement `PercentDone`, l'argument `Percent` contient le pourcentage de tâche réalisé. L'argument `Cancel` peut avoir la valeur **True** pour annuler la méthode ayant déclenché l'événement.

Remarque
Vous pouvez déclarer des arguments d'événement de la même manière que pour les arguments de procédures, avec toutefois les exceptions suivantes : les événements ne peuvent pas avoir d'arguments **Optional**, **ParamArray**, ni de valeur de retour.

L'événement `PercentDone` est déclenché par la méthode `LongTask` de la classe `Widget`. `LongTask` prend en compte deux arguments : la durée pendant laquelle la méthode prétend travailler et l'intervalle minimal requis avant que `LongTask` ne soit suspendue pour déclencher l'événement `PercentDone`.

Pour déclencher l'événement PercentDone

1. Pour simplifier l'accès à la propriété **Timer** utilisée par cette classe, ajoutez une instruction **Imports** en haut de la section des déclarations de votre module de classe, au-dessus de l'instruction `Class Widget`.

   VB

   ```
   Imports Microsoft.VisualBasic.DateAndTime
   ```

2. Ajoutez le code suivant à la classe `Widget` :

   VB

   ```
   Public Sub LongTask(ByVal Duration As Single, _
                       ByVal MinimumInterval As Single)
       Dim Threshold As Single
       Dim Start As Single
       Dim blnCancel As Boolean

       ' The Timer property of the DateAndTime object returns the seconds
       ' and milliseconds that have passed since midnight.
       Start = CSng(Timer)
       Threshold = MinimumInterval

       Do While CSng(Timer) < (Start + Duration)
           ' In a real application, some unit of work would
           ' be done here each time through the loop.
           If CSng(Timer) > (Start + Threshold) Then
               RaiseEvent PercentDone( _
               Threshold / Duration, blnCancel)
               ' Check to see if the operation was canceled.
               If blnCancel Then Exit Sub
               Threshold = Threshold + MinimumInterval
           End If
       Loop
   End Sub
   ```

Lorsque votre application appelle la méthode `LongTask`, la classe `Widget` déclenche l'événement `PercentDone` chaque `MinimumInterval` seconde. Lorsque l'événement est retourné, la méthode `LongTask` contrôle si l'argument `Cancel` avait la valeur **True**.

Quelques mises en garde s'imposent toutefois à ce stade. Pour des raisons de simplicité, la procédure `LongTask` suppose que vous connaissez au préalable le temps que va durer la tâche, ce qui n'est pratiquement jamais le cas. Il peut être difficile de diviser les tâches en segments de même longueur et, la plupart du temps, l'information qui compte le plus aux yeux des utilisateurs est simplement le temps qui s'écoule avant qu'ils ne voient s'afficher un message indiquant que quelque chose se produit.

Il se peut que vous ayez remarqué une autre faille dans cet exemple. La propriété **Timer** retourne le nombre de secondes écoulées depuis minuit, c'est pourquoi l'application se bloque si elle est démarrée juste avant minuit. Une approche plus prudente pour la mesure du temps consisterait à prendre en compte des conditions de limite telles que celle-ci ou éviter purement et simplement qu'elles n'utilisent des propriétés telles que **Now**.

Maintenant que la classe `Widget` peut déclencher des événements, vous pouvez passer à la procédure pas à pas suivante. Procédure pas à pas : gestion des événements explique comment utiliser **WithEvents** pour associer un gestionnaire d'événements à l'événement `PercentDone`.

Voir aussi

Tâches

Procédure pas à pas : gestion des événements

Comment : écrire des gestionnaires d'événements

Référence

Timer, propriété

WithEvents

Now, propriété

Event, instruction

ByVal

ByRef

Imports, instruction

Concepts

Événements et gestionnaires d'événements

Délégués et opérateur AddressOf

AddHandler et RemoveHandler

Procédure pas à pas : gestion des événements

Cette rubrique est la deuxième des deux rubriques consacrées à la manipulation des événements. La première rubrique, Procédure pas à pas : déclaration et déclenchement des événements, explique comment déclarer et déclencher des événements. La présente section utilise le formulaire et la classe de la première procédure pas à pas pour montrer comment gérer les événements lorsqu'ils se produisent.

L'exemple de classe `Widget` utilise des instructions de gestion des événements traditionnelles. Visual Basic fournit d'autres techniques d'utilisation des événements. En guise d'exercice, vous pouvez modifier cet exemple de façon à utiliser des instructions **AddHandler** et **Handles**.

Pour gérer l'événement PercentDone de la classe Widget

- Tapez le code suivant dans `Form1` :

  VB

  ```
  Private WithEvents mWidget As Widget
  Private mblnCancel As Boolean
  ```

  Le mot clé **WithEvents** spécifie que la variable `mWidget` est utilisée pour gérer les événements d'un objet. Pour spécifier le type d'objet, vous devez fournir le nom de la classe à partir de laquelle cet objet sera créé.

  La variable `mWidget` est déclarée dans `Form1` parce que les variables **WithEvents** doivent être de niveau classe. Cela est vrai quel que soit le type de classe dans lequel vous les placez.

  La variable `mblnCancel` est employée pour annuler la méthode `LongTask`.

Écriture de code pour la gestion d'un événement

Dès que vous déclarez une variable à l'aide de **WithEvents**, le nom de la variable apparaît dans la zone de liste déroulante de gauche dans l'**éditeur de code** de la classe. Quand vous sélectionnez `mWidget`, les événements de la classe `Widget` apparaissent dans la zone de liste déroulante de droite. Si vous sélectionnez un événement, la procédure événementielle correspondante s'affiche, avec le préfixe `mWidget` et un caractère de soulignement. Toutes les procédures événementielles associées à une variable **WithEvents** reçoivent comme préfixe le nom de la variable.

Pour gérer un événement

1. Sélectionnez `mWidget` dans la zone de liste déroulante de gauche dans l'**éditeur de code**.
2. Sélectionnez l'événement `PercentDone` dans la zone de liste déroulante de droite. L'**éditeur de code** ouvre la procédure événementielle `mWidget_PercentDone`.

   Remarque

   L'**éditeur de code** est utile, mais pas obligatoire, pour l'insertion de nouveaux gestionnaires d'événements. Dans cette procédure pas à pas, il est plus rapide de copier simplement les gestionnaires d'événements directement dans votre code.

3. Ajoutez le code suivant au gestionnaire d'événements `mWidget_PercentDone` :

   VB

   ```
   Private Sub mWidget_PercentDone( _
       ByVal Percent As Single, _
       ByRef Cancel As Boolean _
   ) Handles mWidget.PercentDone
       lblPercentDone.Text = CInt(100 * Percent) & "%"
       My.Application.DoEvents()
       If mblnCancel Then Cancel = True
   End Sub
   ```

Chaque fois que l'événement `PercentDone` est déclenché, la procédure événementielle affiche le pourcentage réalisé dans un contrôle **Label**. La méthode **DoEvents** permet que l'étiquette soit repeinte et donne également à l'utilisateur la possibilité de cliquer sur le bouton **Annuler**.

4. Ajoutez le code suivant pour le gestionnaire d'événements `Button2_Click` :

   VB

```
Private Sub Button2_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Button2.Click
    mblnCancel = True
End Sub
```

Si l'utilisateur clique sur le bouton **Annuler** pendant l'exécution de la méthode `LongTask`, l'événement `Button2_Click` est exécuté dès que l'instruction **DoEvents** permet le traitement de l'événement. La variable de classe `mblnCancel` prend la valeur **True** ; l'événement `mWidget_PercentDone` la teste ensuite et assigne la valeur **True** à l'argument `ByRef Cancel`.

Association d'une variable WithEvents à un objet

`Form1` est désormais prêt à gérer les événements d'un objet `Widget`. Il ne reste qu'à trouver un objet `Widget`.

Quand vous déclarez une variable **WithEvents** au moment du design, aucun objet n'est associé à celle-ci. Une variable **WithEvents** n'est qu'une variable objet parmi les autres. Vous devez créer un objet et lui assigner une référence avec la variable **WithEvents**.

Pour créer un objet et lui assigner une référence

1. Sélectionnez **(Form1 Events)** dans la zone de liste déroulante de gauche dans l'**éditeur de code**.
2. Sélectionnez l'événement `Load` dans la zone de liste déroulante de droite. L'**éditeur de code** ouvre la procédure événementielle `Form1_Load`.
3. Ajoutez le code suivant pour la procédure événementielle `Form1_Load` afin de créer le `Widget`:

   VB

```
Private Sub Form1_Load( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs _
) Handles MyBase.Load
    mWidget = New Widget
End Sub
```

Lorsque ce code s'exécute, Visual Basic crée un objet `Widget` et associe ses événements aux procédures événementielles de `mWidget`. À partir de ce moment, chaque fois que l'objet `Widget` déclenche son événement `PercentDone`, la procédure événementielle `mWidget_PercentDone` est exécutée.

Pour appeler la méthode LongTask

- Ajoutez le code suivant au gestionnaire d'événements `Button1_Click` :

  VB

```
Private Sub Button1_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Button1.Click
```

```
    mblnCancel = False
    lblPercentDone.Text = "0%"
    lblPercentDone.Refresh()
    mWidget.LongTask(12.2, 0.33)
    If Not mblnCancel Then lblPercentDone.Text = CStr(100) & "%"
End Sub
```

Avant que la méthode `LongTask` ne soit appelée, l'étiquette qui affiche le pourcentage de réalisation doit être initialisée, et l'indicateur de classe **Boolean** utilisé pour l'annulation de la méthode doit avoir la valeur **False**.

La méthode `LongTask` est appelée, la durée de la tâche étant de 12,2 secondes. L'événement `PercentDone` est déclenché tous les tiers de seconde. À chaque déclenchement de l'événement, la procédure événementielle `mWidget_PercentDone` est exécutée.

Une fois la méthode `LongTask` terminée, la variable `mblnCancel` est testée pour déterminer si `LongTask` s'est terminée normalement ou si elle s'est arrêtée parce que `mblnCancel` avait la valeur **True**. Le pourcentage de réalisation de la tâche est mis à jour seulement dans le premier cas.

Pour exécuter le programme

1. Appuyez sur F5 pour passer le projet en mode exécution.
2. Cliquez sur le bouton **Start Task**. Chaque fois que l'événement `PercentDone` est déclenché, l'étiquette est mise à jour de façon à afficher le pourcentage de tâche réalisé.
3. Cliquez sur le bouton **Cancel** pour arrêter la tâche. Notez que l'apparence du bouton **Annuler** ne change pas immédiatement lorsque vous cliquez dessus. L'événement **Click** ne peut pas se produire tant que l'instruction **My.Application.DoEvents** n'autorise pas le traitement de l'événement.

> **Remarque**
>
> La méthode **My.Application.DoEvents** ne traite pas les événements exactement de la même manière que le formulaire. Par exemple, dans cette procédure pas à pas, vous devez cliquer deux fois sur le bouton **Annuler**. Pour permettre au formulaire de gérer les événements directement, vous pouvez utiliser le multithreading. Pour plus d'informations, consultez Multithreading dans Visual Basic.

Il peut se révéler intéressant d'exécuter le programme en appuyant sur F11 pour exécuter le code pas à pas, une ligne à la fois. Vous verrez alors clairement comment l'exécution passe à la méthode `LongTask`, puis repasse brièvement dans `Form1` chaque fois que l'événement `PercentDone` est déclenché.

Que se passerait-il si, une fois l'exécution de retour dans le code de `Form1`, la méthode `LongTask` était de nouveau appelée ? Au pire, il pourrait y avoir un dépassement de capacité de la pile si la méthode `LongTask` était appelée à chaque déclenchement de l'événement.

Vous pouvez faire en sorte que la variable `mWidget` gère des événements d'un autre objet `Widget` en assignant à `mWidget` une référence au nouvel objet `Widget`. En réalité, vous pouvez faire en sorte que le code de la procédure `Button1_Click` effectue cette opération chaque fois que vous cliquez sur ce bouton.

Pour gérer les événements d'un autre objet Widget

- Ajoutez la ligne de code suivante à la procédure `Button1_Click`, juste avant la ligne `mWidget.LongTask(12.2, 0.33)` :

  VB

```
mWidget = New Widget
' Create a new Widget object.
```

Le code ci-dessus crée un nouvel objet `Widget` chaque fois que l'on clique sur le bouton. Dès que la méthode `LongTask` s'achève, la référence à l'objet `Widget` est libérée et l'objet `Widget` est détruit.

Une variable **WithEvents** ne peut contenir qu'une seule référence d'objet à la fois. Dès lors, si vous assignez un autre objet `Widget` à `mWidget`, les événements de l'ancien objet `Widget` ne sont plus gérés. Si `mWidget` est la seule variable objet qui

contient une référence à l'ancien objet `Widget`, ce dernier est détruit. Pour pouvoir gérer les événements de plusieurs objets `Widget`, utilisez l'instruction **AddHandler** qui permet de traiter séparément les événements de chaque objet.

Remarque
Vous pouvez déclarer autant de variables **WithEvents** qu'il en faut, mais les tableaux de variables **WithEvents** ne sont pas p ris en charge.

Voir aussi

Tâches

Procédure pas à pas : déclaration et déclenchement des événements

Comment : écrire des gestionnaires d'événements

Référence

Handles

WithEvents

Concepts

Événements et gestionnaires d'événements

Délégués et opérateur AddressOf

AddHandler et RemoveHandler

Autres ressources

Polymorphisme

Comment : déclarer des événements qui évitent les blocages

Il existe plusieurs cas où il est important qu'un gestionnaire d'événements ne bloque pas les gestionnaires d'événements suivants. Les événements personnalisés permettent à l'événement d'appeler ses gestionnaires d'événements de façon asynchrone.

Par défaut, le champ du magasin de stockage pour une déclaration de l'événement est un délégué multicast qui associe en série tous les gestionnaires d'événements. Cela signifie que si un gestionnaire met un certain temps à s'exécuter, les autres gestionnaires sont bloqués tant qu'il n'a pas terminé. (Les gestionnaires d'événements valides ne doivent jamais exécuter des opérations longues ou susceptibles de créer un blocage).

Au lieu d'utiliser l'implémentation d'événements par défaut fournie par Visual Basic, vous pouvez utiliser un événement personnalisé pour exécuter les gestionnaires d'événements de façon asynchrone.

Exemple

Dans cet exemple, l'accesseur **AddHandler** ajoute le délégué pour chaque gestionnaire de l'événement `Click` dans un ArrayList stocké dans le champ `EventHandlerList`.

Lorsque le code déclenche l'événement `Click`, l'accesseur **RaiseEvent** appelle tous les délégués de gestionnaire d'événements de façon asynchrone à l'aide de la méthode BeginInvoke. Étant donné que cette méthode appelle chaque gestionnaire sur un thread de travail et retourne immédiatement, les gestionnaires ne peuvent pas se bloquer mutuellement.

VB

```
Public NotInheritable Class ReliabilityOptimizedControl
    'Defines a list for storing the delegates
    Private EventHandlerList As New ArrayList

    'Defines the Click event using the custom event syntax.
    'The RaiseEvent always invokes the delegates asynchronously
    Public Custom Event Click As EventHandler
        AddHandler(ByVal value As EventHandler)
            EventHandlerList.Add(value)
        End AddHandler
        RemoveHandler(ByVal value As EventHandler)
            EventHandlerList.Remove(value)
        End RemoveHandler
        RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
            For Each handler As EventHandler In EventHandlerList
                If handler IsNot Nothing Then
                    handler.BeginInvoke(sender, e, Nothing, Nothing)
                End If
            Next
        End RaiseEvent
    End Event
End Class
```

Voir aussi

Tâches

Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Référence

Event, instruction

ArrayList

BeginInvoke

Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Il existe plusieurs situations où il est important pour une application d'économiser l'utilisation de sa mémoire. Les événements personnalisés permettent à l'application d'utiliser la mémoire uniquement pour les événements qu'elle gère.

Par défaut, lorsqu'une classe déclare un événement, le compilateur alloue de la mémoire à un champ pour stocker les informations sur l'événement. Si une classe contient plusieurs événements inutilisés, ceux-ci utilisent de la mémoire inutilement.

Au lieu d'utiliser l'implémentation d'événements par défaut fournie par Visual Basic, vous pouvez utiliser des événements personnalisés pour gérer plus consciencieusement l'utilisation de la mémoire.

Exemple

Dans cet exemple, la classe utilise une instance de la classe EventHandlerList, qui est contenue dans le champ `Events`, pour stocker les informations sur les événements utilisés. La classe **EventHandlerList** est une classe de liste optimisée conçue pour contenir des délégués.

Tous les événements de la classe utilisent le champ `Events` pour assurer le suivi des méthodes qui gèrent chaque événement.

VB

```
Public Class MemoryOptimizedBaseControl
    ' Define a delegate store for all event handlers.
    Private Events As New System.ComponentModel.EventHandlerList

    ' Define the Click event to use the delegate store.
    Public Custom Event Click As EventHandler
        AddHandler(ByVal value As EventHandler)
            Events.AddHandler("ClickEvent", value)
        End AddHandler
        RemoveHandler(ByVal value As EventHandler)
            Events.RemoveHandler("ClickEvent", value)
        End RemoveHandler
        RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
            CType(Events("ClickEvent"), EventHandler).Invoke(sender, e)
        End RaiseEvent
    End Event

    ' Define the Click event to use the same delegate store.
    Public Custom Event DoubleClick As EventHandler
        AddHandler(ByVal value As EventHandler)
            Events.AddHandler("DoubleClickEvent", value)
        End AddHandler
        RemoveHandler(ByVal value As EventHandler)
            Events.RemoveHandler("DoubleClickEvent", value)
        End RemoveHandler
        RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
            CType(Events("DoubleClickEvent"), EventHandler).Invoke(sender, e)
        End RaiseEvent
    End Event

    ' Define additional events to use the same delegate store.
    ' ...
End Class
```

Voir aussi

Tâches

Comment : déclarer des événements qui évitent les blocages

Référence

Event, instruction

EventHandlerList

Dépannage des gestionnaires d'événements hérités en Visual Basic .2005

Cette rubrique décrit les problèmes courants relatifs aux gestionnaires d'événements dans les composants hérités.

Procédures

Le code d'un gestionnaire d'événements s'exécute deux fois pour chaque appel

- Un gestionnaire d'événements hérité ne doit pas inclure de clause Handles. La méthode de la classe de base est déjà associée à l'événement et se déclenche en conséquence. Supprimez la clause **Handles** de la méthode héritée.

  VB

```
' INCORRECT
Protected Overrides Sub Button1_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click

    ' The Handles clause will cause all code
    ' in this block to be executed twice.
End Sub
```

- Si la méthode héritée ne comporte pas de mot clé **Handles**, vérifiez que votre code ne contient pas d'instruction AddHandler supplémentaire ou d'autres méthodes gérant le même événement.

Voir aussi

Tâches

Comment : écrire des gestionnaires d'événements

Concepts

Événements et gestionnaires d'événements

Comment : gérer les événements dans Visual Basic

Autres ressources

Événements en Visual Basic

Délégués en Visual Basic

Un délégué est une forme de pointeur fonction orienté objet permettant à une fonction d'être appelée indirectement par l'intermédiaire d'une référence à la fonction. Les délégués peuvent être utilisés pour raccorder des gestionnaires d'événements et passer d'une procédure à une autre.

Dans cette section

Délégués et opérateur AddressOf

Offre une introduction aux délégués et à leur utilisation.

Comment : appeler une méthode déléguée

Crée une procédure de tri dans une classe qui utilise le tri alphabétique standard dans la plupart des zones de liste, mais qui est capable, au moment de l'exécution, de passer à la procédure de tri personnalisée. Pour ce faire, vous faites appel à des délégués pour passer la procédure de tri personnalisée à la classe de tri au moment de l'exécution.

Comment : passer des procédures à une autre procédure dans Visual Basic

Illustre comment utiliser des délégués pour passer d'une procédure à une autre.

Rubriques connexes

Événements en Visual Basic

Vue d'ensemble d'événements en Visual Basic.

Gestion et déclenchement d'événements

Propose une vue d'ensemble du modèle d'événement dans le .NET Framework.

Création de gestionnaires d'événements dans les Windows Forms

Décrit comment utiliser des événements associés aux objets Windows Forms.

Délégués et opérateur AddressOf

Les délégués sont des objets que vous pouvez utiliser pour appeler les méthodes d'autres objets. Ils sont parfois décrits comme des *pointeurs fonction de type sécurisé* car ils sont comparables aux pointeurs fonction utilisés dans d'autres langages de programmation. Mais contrairement aux pointeurs fonction, les délégués Visual Basic sont un type référence basé sur la classe System.Delegate. Les délégués peuvent faire référence à des méthodes partagées (méthodes qui peuvent être appelées sans une instance spécifique d'une classe) et à des méthodes d'instance.

Délégués et événements

Les délégués sont utiles lorsque vous avez besoin d'un intermédiaire entre une procédure appelante et la procédure appelée. Vous pouvez, par exemple, souhaiter un objet qui déclenche des événements pour pouvoir appeler différents gestionnaires d'événements dans des circonstances différentes. Malheureusement, l'objet qui génère les événements ne peut pas savoir au préalable quel gestionnaire d'événements gère un événement spécifique. Visual Basic vous laisse associer de manière dynamique des gestionnaires d'événements à des événements en créant un délégué à votre intention lorsque vous utilisez l'instruction **AddHandler**. Au moment de l'exécution, le délégué transmet les appels au gestionnaire d'événements approprié.

Bien que vous puissiez créer vos propres délégués, c'est Visual Basic qui se charge le plus souvent de créer le délégué et de gérer tous les détails à votre place. Par exemple, une instruction **Event** définit implicitement une classe déléguée appelée `<EventName>EventHandler` en tant que classe imbriquée de la classe contenant l'instruction **Event**, ce avec la même signature que l'événement. L'instruction **AddressOf** crée implicitement une instance de délégué. Par exemple, les deux lignes de code suivantes sont équivalentes :

VB

```
AddHandler Button1.Click, AddressOf Me.Button1_Click
' The previous line of code is shorthand for the next line of code.
AddHandler Button1.Click, New EventHandler(AddressOf Button1_Click)
```

Vous pouvez utiliser la méthode abrégée de création de délégués à chaque fois que le compilateur peut déterminer le type du délégué en fonction du contexte.

Déclaration d'événements utilisant un type délégué existant

Dans certaines situations, vous pouvez choisir de déclarer un événement pour utiliser un type délégué existant comme son délégué sous-jacent. La syntaxe ci-dessous vous indique comment procéder :

VB

```
Delegate Sub DelegateType()
Event AnEvent As DelegateType
```

Cela peut vous être utile lorsque vous voulez router plusieurs événements vers le même gestionnaire.

Variables et paramètres de délégués

Vous pouvez utiliser des délégués pour d'autres tâches non liées à des événements, par exemple pour le traitement dissocié des threads ou pour des procédures impliquant d'appeler différentes versions de fonctions au moment de la compilation.

Supposons, par exemple, que vous ayez une application de gestion de petites annonces contenant une zone de liste affichant les noms de voitures. Les annonces sont triées par titre, lequel correspond normalement à la marque de la voiture. Un problème peut néanmoins se poser lorsque, pour certaines voitures, la marque est précédée de l'année de mise en circulation. Ce cas représente un vrai problème, car la fonctionnalité de tri intégrée de la zone de liste base son tri sur les codes de caractères ; elle commence donc par trier en tête de liste toutes les annonces dont l'intitulé commence par des dates pour ensuite traiter les annonces dont l'intitulé commence par la marque.

Pour résoudre ce problème, vous pouvez créer une procédure de tri dans une classe qui utilise le tri alphabétique standard dans la plupart des zones de liste, mais qui est capable, au moment de l'exécution, de passer à la procédure de tri personnalisée définie pour les petites annonces concernant les voitures. Pour ce faire, vous faites appel à des délégués pour passer la procédure de tri personnalisée à la classe de tri au moment de l'exécution.

Chaque classe déléguée définit un constructeur qui prend une spécification d'une méthode objet. La syntaxe ci-dessous fournit

un exemple de paramètre pour un constructeur délégué de ce type :

AddressOf [*expression*.]*methodname*

Le type de *expression* au moment de la compilation doit correspondre à une classe ou à une interface avec une méthode portant le nom spécifié dont la signature correspond à la signature de la classe déléguée. La méthode peut être soit une méthode (classe) partagée, soit une méthode d'instance.

Voir aussi

Tâches

Comment : passer des procédures à une autre procédure dans Visual Basic
Comment : appeler une méthode déléguée
Comment : écrire des gestionnaires d'événements

Référence

Delegate, instruction
AddressOf, opérateur

Concepts

Événements et gestionnaires d'événements
AddHandler et RemoveHandler
Applications multithread

Comment : appeler une méthode déléguée

Cet exemple montre comment associer une méthode à un délégué, puis comment appeler cette méthode par le biais du délégué.

Création du délégué et des procédures correspondantes

1. Créez un délégué nommé `MySubDelegate`.

```
Delegate Sub MySubDelegate(ByVal x As Integer)
```

2. Déclarez une classe qui contient une méthode avec la même signature que le délégué.

```
Class class1
    Sub Sub1(ByVal x As Integer)
        MsgBox("The value of x is: " & CStr(x))
    End Sub
End Class
```

3. Définissez une méthode qui crée une instance du délégué et appelle la méthode associée au délégué en appelant la méthode **Invoke** intégrée.

```
Protected Sub DelegateTest()
    Dim c1 As New class1
    ' Create an instance of the delegate.
    Dim msd As MySubDelegate = AddressOf c1.Sub1
    ' Call the method.
    msd.Invoke(10)
End Sub
```

Voir aussi

Tâches

Comment : écrire des gestionnaires d'événements

Référence

Delegate, instruction

Concepts

Délégués et opérateur AddressOf

Événements et gestionnaires d'événements

AddHandler et RemoveHandler

Applications multithread

Comment : passer des procédures à une autre procédure dans Visual Basic

Cet exemple montre comment utiliser des délégués pour passer une procédure à une autre procédure.

Un délégué est un type que vous pouvez utiliser comme tout autre type dans Visual Basic. L'opérateur **AddressOf** retourne un objet délégué lorsqu'il est appliqué à un nom de procédure.

Cet exemple comporte une procédure avec un paramètre de délégué qui peut prendre une référence à une autre procédure, obtenue avec l'opérateur **AddressOf**.

Création du délégué et des procédures correspondantes

1. Créez un délégué nommé `MathOperator`.

   VB

   ```
   Delegate Function MathOperator( _
       ByVal x As Double, _
       ByVal y As Double _
   ) As Double
   ```

2. Créez une procédure nommée `AddNumbers` avec des paramètres et une valeur de retour qui correspondent à ceux de `MathOperator`, de sorte que les signatures correspondent.

   VB

   ```
   Function AddNumbers( _
       ByVal x As Double, _
       ByVal y As Double _
   ) As Double
       Return x + y
   End Function
   ```

3. Créez une procédure nommée `SubtractNumbers` avec une signature qui correspond à `MathOperator`.

   VB

   ```
   Function SubtractNumbers( _
       ByVal x As Double, _
       ByVal y As Double _
   ) As Double
       Return x - y
   End Function
   ```

4. Créez une procédure nommée `DelegateTest` qui prend un délégué comme paramètre.

   Cette procédure peut accepter une référence à `AddNumbers` ou `SubtactNumbers`, car leurs signatures correspondent à la signature de `MathOperator`.

   VB

   ```
   Sub DelegateTest( _
       ByVal x As Double, _
       ByVal op As MathOperator, _
       ByVal y As Double _
   )
   ```

```
    Dim ret As Double
    ret = op.Invoke(x, y) ' Call the method.
    MsgBox(ret)
End Sub
```

5. Créez une procédure nommée `Test` qui appelle `DelegateTest` une fois avec le délégué de `AddNumbers` comme paramètre, puis une autre fois avec le délégué de `SubtractNumbers` comme paramètre.

   VB

```
Protected Sub Test()
    DelegateTest(5, AddressOf AddNumbers, 3)
    DelegateTest(9, AddressOf SubtractNumbers, 3)
End Sub
```

   Lorsque la procédure `Test` est appelée, elle affiche d'abord le résultat de l'action de `AddNumbers` sur `5` et `3`, qui est égal à 8. Ensuite, le résultat de l'action de `SubtractNumbers` sur `9` et `3` est affiché, qui est égal à 6.

Voir aussi

Tâches

Comment : appeler une méthode déléguée

Référence

AddressOf, opérateur

Delegate, instruction

Autres ressources

Délégués en Visual Basic

Interfaces en Visual Basic

Les *interfaces* définissent les propriétés, méthodes et événements qui peuvent être implémentés par les classes. Les interfaces vous permettent de définir des fonctionnalités sous forme de petits groupes de propriétés, méthodes et événements étroitement liés ; les risques d'incompatibilité s'en trouvent limités parce que vous avez la possibilité de développer des implémentations avancées sans détruire le code existant. Vous pouvez ajouter de nouvelles fonctionnalités à tout moment en développant des interfaces et implémentations supplémentaires.

Dans les versions antérieures de Visual Basic, vous pouviez consommer des interfaces, mais pas les créer directement. Dans la présente version, vous pouvez utiliser l'instruction **Interface**, qui permet de définir de vraies interfaces en tant qu'entités distinctes des classes et de les implémenter avec une version améliorée du mot clé **Implements**.

Dans cette section

Vue d'ensemble des interfaces

Fournit une vue d'ensemble de ce que sont les interfaces et de la manière dont Visual Basic les implémente.

Définition des interfaces

Décrit comment définir les interfaces avec les instructions **Interface** et **End Interface**.

Implements, mot clé et instruction

Décrit comment indiquer qu'une section de code implémente une interface spécifique.

Exemples d'implémentation d'interface dans Visual Basic

Fournit trois exemples d'implémentation d'interface.

Quand utiliser des interfaces

Explique dans quels cas il convient d'utiliser des interfaces au lieu d'une hiérarchie d'héritage.

Comment : créer et implémenter des interfaces

Répertorie les étapes intervenant dans la définition et l'implémentation d'une interface et en fournit des exemples.

Procédure pas à pas : création et implémentation d'interfaces

Fournit une procédure détaillée qui vous fait découvrir le processus de définition et d'implémentation de votre propre interface.

Rubriques connexes

Héritage en Visual Basic

Décrit comment Visual Basic prend en charge l'héritage, ce qui vous permet de définir les classes qui servent de base pour les classes dérivées.

Vue d'ensemble des interfaces

Tout comme les classes, les interfaces définissent un ensemble de propriétés, méthodes et événements. Cependant, contrairement aux classes, les interfaces n'assurent pas l'implémentation. Elles sont implémentées par les classes et définies en tant qu'entités distinctes des classes.

Une interface représente un contrat, dans le sens où une classe qui implémente une interface doit implémenter tous les aspects de cette interface exactement telle qu'elle a été définie.

Avec les interfaces, vous pouvez définir les fonctionnalités en tant que petits groupes de membres étroitement liés. Vous pouvez développer des implémentations avancées pour vos interfaces sans endommager le code existant, ce qui limite les problèmes de compatibilité. Vous pouvez également ajouter de nouvelles fonctionnalités à tout moment en développant des interfaces et implémentations supplémentaires.

Même si les implémentations d'interfaces peuvent évoluer, les interfaces elles-mêmes ne peuvent pas être modifiées après publication. En effet, toute modification apportée à une interface publiée risque d'endommager le code existant. Si vous partez du principe qu'une interface est une sorte de contrat, il apparaît clairement que les deux parties liées par ce contrat ont un rôle à jouer. La partie qui publie une interface accepte de ne jamais modifier cette dernière et l'implémenteur accepte de l'implémenter exactement comme elle a été conçue.

Dans les versions antérieures de Visual Basic, vous pouviez consommer des interfaces, mais pas les créer directement. Désormais, vous pouvez définir de vraies interfaces à l'aide de l'instruction **Interface** et vous pouvez implémenter des interfaces avec une version améliorée du mot clé **Implements**.

Voir aussi

Tâches

Comment : créer et implémenter des interfaces
Procédure pas à pas : création et implémentation d'interfaces

Référence

Interface, instruction (Visual Basic)

Concepts

Définition des interfaces
Implements, mot clé et instruction
Exemples d'implémentation d'interface dans Visual Basic
Quand utiliser des interfaces

Autres ressources

Héritage en Visual Basic

Définition des interfaces

Les définitions d'interfaces sont comprises dans les instructions **Interface** et **End Interface**. À la suite de l'instruction **Interface**, vous pouvez ajouter une instruction **Inherits** facultative qui répertorie une ou plusieurs interfaces héritées. Les instructions **Inherits** doivent précéder toutes les autres instructions dans la déclaration, à l'exception des commentaires. Les autres instructions figurant dans la définition de l'interface doivent être **Event**, **Sub**, **Function**, **Property**, **Interface**, **Class**, **Structure** et **Enum**. Les interfaces ne peuvent pas contenir de code d'implémentation ni d'instructions associées au code d'implémentation, telles que **End Sub** ou **End Property**.

Dans un espace de noms, les instructions d'interface sont par défaut **Friend**, mais elles peuvent également être déclarées explicitement comme **Public** ou **Friend**. Les interfaces définies dans des classes, des modules, des interfaces et des structures sont par défaut **Public**, mais elles peuvent également être déclarées explicitement comme **Public**, **Friend**, **Protected** ou **Private**.

Remarque

Le mot clé **Shadows** peut être appliqué à tous les membres d'interface. Le mot clé **Overloads** peut être appliqué aux instructions **Sub**, **Function** et **Property** déclarées dans une définition d'interface. Par ailleurs, les instructions **Property** peuvent posséder les modificateurs **Default**, **ReadOnly** ou **WriteOnly**. Aucun autre modificateur — **Public**, **Private**, **Friend**, **Protected**, **Shared**, **Overrides**, **MustOverride** ou **Overridable** — n'est autorisé. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Voir aussi

Tâches

Comment : créer et implémenter des interfaces
Procédure pas à pas : création et implémentation d'interfaces

Référence

Interface, instruction (Visual Basic)
Inherits, instruction
Overloads
Default (Visual Basic)

Concepts

Vue d'ensemble des interfaces
Implements, mot clé et instruction
Exemples d'implémentation d'interface dans Visual Basic
Quand utiliser des interfaces

Autres ressources

Héritage en Visual Basic

Implements, mot clé et instruction

Le mot réservé Visual Basic **Implements** est utilisé de deux manières. L'instruction **Implements** signifie qu'une classe ou une structure implémente une interface. Le mot clé **Implements** signifie qu'un membre de classe ou un membre de structure implémente un membre d'interface spécifique.

Instruction d'implémentation

Si une classe ou une structure implémente une ou plusieurs interfaces, elle doit inclure l'instruction **Implements** juste après l'instruction **Class** ou **Structure**. L'implémentation de l'instruction **Implements** par une classe suppose l'existence d'une liste d'interfaces séparées par des virgules. La classe ou structure doit implémenter tous les membres d'interface à l'aide du mot clé **Implements**.

Implements, mot clé

L'implémentation du mot clé **Implements** suppose l'existence d'une liste de membres d'interfaces séparés par des virgules. En général, un seul membre d'interface est spécifié, mais vous pouvez tout à fait spécifier plusieurs membres. La spécification d'un membre d'interface se présente sous la forme du nom de l'interface (qui doit être spécifié dans une instruction implements dans la classe), suivi d'un point et du nom de la fonction membre, de la propriété ou de l'événement à implémenter. Le nom d'un membre qui implémente un membre d'interface peut utiliser n'importe quel identificateur conforme et ne doit pas obligatoirement respecter strictement la convention **InterfaceName_MethodName** utilisée dans les versions antérieures de Visual Basic.

Par exemple, le code ci-dessous indique comment déclarer une sous-routine nommée `Sub1` qui implémente une méthode d'une interface :

VB

```
Class Class1
    Implements interfaceclass.interface2

    Sub Sub1(ByVal i As Integer) Implements interfaceclass.interface2.Sub1
    End Sub
End Class
```

Les types de paramètres et types de retour du membre qui implémente doivent correspondre à la propriété d'interface ou à la déclaration de membre dans l'interface. La façon la plus répandue d'implémenter un élément d'interface consiste à utiliser un membre portant le même nom que l'interface, comme dans l'exemple précédent.

Pour déclarer l'implémentation d'une méthode d'interface, vous pouvez utiliser tous les attributs conformes dans les déclarations de méthodes d'instances, notamment **Overloads**, **Overrides**, **Overridable**, **Public**, **Private**, **Protected**, **Friend**, **Protected Friend**, **MustOverride**, **Default** et **Static**. L'attribut **Shared** n'est pas conforme parce qu'il définit une classe au lieu d'une méthode d'instance.

Implements permet également d'écrire une seule méthode implémentant plusieurs méthodes définies dans une interface, comme dans l'exemple suivant :

VB

```
Class Class2
    Implements I1, I2

    Protected Sub M1() Implements I1.M1, I1.M2, I2.M3, I2.M4
    End Sub
End Class
```

Vous pouvez utiliser un membre privé pour implémenter un membre d'interface. Lorsqu'un membre privé implémente un membre d'interface, ce membre devient disponible via l'interface, même s'il n'est pas directement disponible avec les variables objets pour la classe.

Voir aussi

Tâches

Comment : créer et implémenter des interfaces
Procédure pas à pas : création et implémentation d'interfaces

Référence

Implements, instruction
Implements (Visual Basic)

Concepts

Vue d'ensemble des interfaces
Définition des interfaces
Exemples d'implémentation d'interface dans Visual Basic
Quand utiliser des interfaces

Autres ressources

Héritage en Visual Basic

Exemples d'implémentation d'interface dans Visual Basic

Les classes qui implémentent une interface doivent implémenter toutes ses propriétés, méthodes et événements.

L'exemple à suivre définit deux interfaces. La seconde interface, `Interface2`, hérite de `Interface1` et définit une propriété et une méthode supplémentaires.

VB

```
Interface Interface1
    Sub sub1(ByVal i As Integer)
End Interface

' Demonstrates interface inheritance.
Interface Interface2
    Inherits Interface1
    Sub M1(ByVal y As Integer)
    ReadOnly Property Num() As Integer
End Interface
```

L'exemple suivant implémente `Interface1`, l'interface définie dans l'exemple précédent :

VB

```
Public Class ImplementationClass1
    Implements Interface1
    Sub Sub1(ByVal i As Integer) Implements Interface1.sub1
        ' Insert code here to implement this method.
    End Sub
End Class
```

Ce dernier exemple implémente `Interface2`, y compris une méthode héritée de `Interface1` :

VB

```
Public Class ImplementationClass2
    Implements Interface2
    Dim INum As Integer = 0
    Sub sub1(ByVal i As Integer) Implements Interface2.sub1
        ' Insert code here that implements this method.
    End Sub
    Sub M1(ByVal x As Integer) Implements Interface2.M1
        ' Insert code here to implement this method.
    End Sub

    ReadOnly Property Num() As Integer Implements _
       Interface2.Num
        Get
            Num = INum
        End Get
    End Property
End Class
```

Voir aussi

Tâches

Comment : créer et implémenter des interfaces

Procédure pas à pas : création et implémentation d'interfaces

Référence

Interface, instruction (Visual Basic)

Implements, instruction

Concepts

Vue d'ensemble des interfaces

Définition des interfaces
Implements, mot clé et instruction
Quand utiliser des interfaces

Quand utiliser des interfaces

Les interfaces sont un outil de programmation puissant parce qu'elles vous permettent de séparer la définition d'objets de leur implémentation. Les interfaces et l'héritage de classes présentent chacun des avantages et des inconvénients, ce qui peut vous amener à associer l'utilisation des deux dans vos projets. Cette page et Quand utiliser l'héritage vous aideront à déterminer quelle est la meilleure approche pour votre situation.

Souplesse d'implémentation

Il existe plusieurs autres raisons qui peuvent vous amener à préférer les interfaces à l'héritage de classes :

- Les interfaces sont mieux adaptées aux applications exigeant la création de nombreux types d'objets pas nécessairement liés pour assurer certaines fonctionnalités.
- Les interfaces sont plus souples à utiliser que les classes de base, car il suffit de définir une seule implémentation pour pouvoir implémenter plusieurs interfaces.
- Les interfaces sont plus indiquées lorsque vous n'avez pas besoin d'hériter une implémentation d'une classe de base.
- Les interfaces sont utiles lorsque vous ne pouvez pas utiliser l'héritage de classes. Par exemple, les structures ne peuvent pas hériter des classes, mais elles peuvent implémenter des interfaces.

Voir aussi

Tâches

Comment : créer et implémenter des interfaces
Procédure pas à pas : création et implémentation d'interfaces

Concepts

Quand utiliser l'héritage
Vue d'ensemble des interfaces
Définition des interfaces
Implements, mot clé et instruction
Exemples d'implémentation d'interface dans Visual Basic

Autres ressources

Héritage en Visual Basic

Comment : créer et implémenter des interfaces

Conformément à ce qui est expliqué dans Vue d'ensemble des interfaces, les interfaces décrivent les propriétés, les méthodes et les événements d'une classe sans en assurer l'implémentation.

Pour créer une interface

1. Définissez votre interface en y ajoutant du code commençant par le mot clé **Interface** et le nom de l'interface et se terminant par l'instruction **End Interface**. Par exemple, le code suivant définit une interface appelée `IAsset`.

   VB

   ```
   Interface IAsset
   End Interface
   ```

2. Ajoutez des instructions définissant les propriétés, méthodes et événements pris en charge par votre interface. Par exemple, le code suivant définit une fonction, une propriété et un événement.

   VB

   ```
   Interface IAsset
       Event ComittedChange(ByVal Success As Boolean)
       Property Division() As String
       Function GetID() As Integer
   End Interface
   ```

Pour implémenter une interface

1. Si l'interface que vous implémentez ne fait pas partie de votre projet, ajoutez une référence à l'assembly qui contient l'interface.
2. Créez une nouvelle classe qui implémente votre interface et ajoutez le mot clé **Implements** dans la ligne à la suite du nom de la classe. Par exemple, pour implémenter l'interface `IAsset`, vous pouvez nommer la classe d'implémentation `Computer`, comme dans le code suivant.

   VB

   ```
   Class Computer
       Implements IAsset
   End Class
   ```

3. Ajoutez des procédures pour implémenter les propriétés, méthodes et événements de la classe conformément au code ci-dessous, qui complète l'exemple fourni à l'étape précédente :

   VB

   ```
   Class Computer
       Implements IAsset

       Public Event ComittedChange(ByVal Success As Boolean) _
          Implements IAsset.ComittedChange

       Private divisionValue As String

       Public Property Division() As String _
           Implements IAsset.Division
   ```

```
        Get
            Return divisionValue
        End Get
        Set(ByVal value As String)
            divisionValue = value
            RaiseEvent ComittedChange(True)
        End Set
    End Property

    Private IDValue As Integer

    Public Function GetID() As Integer _
        Implements IAsset.GetID

        Return IDValue
    End Function

    Public Sub New(ByVal Division As String, ByVal ID As Integer)
        Me.divisionValue = Division
        Me.IDValue = ID
    End Sub
End Class
```

Voir aussi

Tâches

Procédure pas à pas : création et implémentation d'interfaces

Référence

Interface, instruction (Visual Basic)

Concepts

Vue d'ensemble des interfaces

Définition des interfaces

Implements, mot clé et instruction

Exemples d'implémentation d'interface dans Visual Basic

Quand utiliser des interfaces

Autres ressources

Héritage en Visual Basic

Procédure pas à pas : création et implémentation d'interfaces

Les interfaces décrivent les caractéristiques des propriétés, des méthodes et des événements, mais laissent les détails d'implémentation aux structures ou aux classes.

Cette procédure pas à pas explique comment déclarer et implémenter une interface.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour définir une interface

1. Ouvrez un nouveau projet d'application Windows Visual Basic.
2. Ajoutez un nouveau module au projet en cliquant sur **Ajouter un module** dans le menu **Projet**.
3. Nommez le nouveau module `Module1.vb` et cliquez sur **Ajouter**. Le code du nouveau module s'affiche.
4. Définissez une interface nommée `TestInterface` dans `Module1` en tapant `Interface TestInterface` entre les instructions **Module** et **End Module**, puis en appuyant sur ENTRÉE. L'**éditeur de code** met en retrait le mot clé **Interface** et ajoute une instruction **End Interface** pour constituer un bloc de code.
5. Définissez une propriété, une méthode et un événement pour l'interface en insérant le code suivant entre les instructions **Interface** et **End Interface** :

   VB

   ```
   Property Prop1() As Integer
   Sub Method1(ByVal X As Integer)
   Event Event1()
   ```

Implémentation

Vous remarquez peut-être que la syntaxe utilisée pour déclarer des membres d'interface est différente de celle utilisée pour déclarer des membres de classe. Cette différence reflète le fait que les interfaces ne peuvent pas contenir le code d'implémentation.

Pour implémenter l'interface

1. Ajoutez une classe nommée `ImplementationClass` en ajoutant l'instruction ci-après à `Module1` entre les instructions `End Interface` et `End Module`, puis appuyez sur ENTRÉE :

   VB

   ```
   Class ImplementationClass
   ```

   Si vous travaillez dans l'environnement de développement intégré, l'**éditeur de code** fournit une instruction `End Class` concordante lorsque vous appuyez sur ENTRÉE.

2. Ajoutez l'instruction **Implements** suivante à `ImplementationClass`, qui nomme l'interface que la classe implémente :

   VB

   ```
   Implements TestInterface
   ```

   Lorsqu'elle est séparée d'autres éléments en haut d'une classe ou d'une structure, l'instruction **Implements** indique que

la classe ou structure implémente une interface.

Si vous travaillez dans l'environnement de développement intégré, l'**éditeur de code** implémente les membres de classe requis par `TestInterface` lorsque vous appuyez sur ENTRÉE et vous pouvez ignorer l'étape suivante.

3. Si vous ne travaillez pas dans l'environnement de développement intégré, vous devez implémenter tous les membres de l'interface `MyInterface`. Ajoutez le code suivant à `ImplementationClass` pour implémenter `Event1`, `Method1` et `Prop1` :

   VB

   ```
   Event Event1() Implements TestInterface.Event1

   Public Sub Method1(ByVal X As Integer) Implements TestInterface.Method1
   End Sub

   Public Property Prop1() As Integer Implements TestInterface.Prop1
       Get
       End Get
       Set(ByVal value As Integer)
       End Set
   End Property
   ```

   L'instruction **Implements** nomme l'interface et le membre d'interface en cours d'implémentation.

4. Complétez la définition de `Prop1` en ajoutant un champ privé à la classe qui a stocké la valeur de propriété :

   VB

   ```
   ' Holds the value of the property.
   Private pval As Integer
   ```

   Retournez la valeur du `pval` de l'accesseur get de propriété.

   VB

   ```
   Return pval
   ```

   Définissez la valeur de `pval` dans l'accesseur set de propriété.

   VB

   ```
   pval = value
   ```

5. Complétez la définition de `Method1` en ajoutant le code suivant.

   VB

   ```
   MsgBox("The X parameter for Method1 is " & X)
   RaiseEvent Event1()
   ```

Pour tester l'implémentation de l'interface

1. Cliquez avec le bouton droit sur le formulaire de démarrage de votre projet dans l'**Explorateur de solutions** et cliquez sur **Afficher le code**. L'éditeur affiche la classe de votre formulaire de départ. Par défaut, le formulaire de démarrage s'appelle `Form1`.

2. Ajoutez le champ suivant `testInstance` à la classe `Form1` :

VB

```
Dim WithEvents testInstance As TestInterface
```

En déclarant `testInstance` comme **WithEvents**, la classe `Form1` peut gérer ses événements.

3. Ajoutez le gestionnaire d'événements suivant à la classe `Form1` pour gérer les événements déclenchés par `testInstance` :

VB

```
Sub EventHandler() Handles testInstance.Event1
    MsgBox("The event handler caught the event.")
End Sub
```

4. Ajoutez une sous-routine nommée `Test` à la classe `Form1` pour tester la classe d'implémentation :

VB

```
Sub Test()
    '  Create an instance of the class.
    Dim T As New ImplementationClass
    ' Assign the class instance to the interface.
    ' Calls to the interface members are
    ' executed through the class instance.
    testInstance = T
    ' Set a property.
    testInstance.Prop1 = 9
    ' Read the property.
    MsgBox("Prop1 was set to " & testInstance.Prop1)
    '  Test the method and raise an event.
    testInstance.Method1(5)
End Sub
```

La procédure `Test` crée une instance de la classe qui implémente `MyInterface`, assigne cette instance au champ `testInstance`, définit une propriété et exécute une méthode par le biais de l'interface.

5. Ajoutez du code pour appeler la procédure `Test` à partir de la procédure `Form1 Load` de votre formulaire de départ :

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Test() ' Test the class.
End Sub
```

6. Exécutez la procédure `Test` en appuyant sur F5. Le message "Prop1 was set to 9" s'affiche. Après avoir cliqué sur OK, le message "The X parameter for Method1 is 5" s'affiche. Cliquez sur OK et le message "The event handler caught the event" s'affiche.

Voir aussi

Tâches

Comment : créer et implémenter des interfaces

Référence

Implements, instruction

Interface, instruction (Visual Basic)

Event, instruction

Concepts

Quand utiliser des interfaces

Autres ressources

Interfaces en Visual Basic

Héritage en Visual Basic

Visual Basic prend en charge l'*héritage*, la capacité de définir les classes qui constitueront la base des classes dérivées. Les classes dérivées héritent des propriétés, des méthodes et des événements de la classe de base et peuvent les étendre. Elles peuvent également substituer des méthodes héritées par de nouvelles implémentations. Toutes les classes créées avec Visual Basic peuvent être héritées par défaut.

L'héritage vous permet d'écrire et de déboguer une seule fois une classe et de réutiliser ensuite ce code plusieurs fois comme base des nouvelles classes. L'héritage vous permet également d'utiliser le *polymorphisme* par héritage, permettant de définir des classes pouvant être utilisées de manière interchangeable par le code client au moment de l'exécution, avec des méthodes et des propriétés différentes d'un point de vue fonctionnel, mais portant le même nom.

Dans cette section

Éléments fondamentaux de l'héritage

Présente les modificateurs d'héritage, la substitution de méthodes et de propriétés, **MyClass** et **MyBase**.

Comment : créer des classes dérivées

Décrit étape par étape le processus de création des classes dérivées.

Quand utiliser l'héritage

Explique dans quels cas il convient d'utiliser l'héritage au lieu des interfaces.

Héritage et le .NET Framework

Explique comment le .NET Framework prend en charge l'héritage.

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Décrit l'utilisation d'objets COM existants en tant que base des nouveaux objets.

Héritage et la classe Object de base

Propose une vue d'ensemble de la classe servant de base pour toutes les autres classes.

Fonctionnement des méthodes New et Finalize dans une hiérarchie de classe

Décrit l'appel aux constructeurs et aux destructeurs dans une hiérarchie de classe.

Polymorphisme

Présente le polymorphisme et son utilisation.

Design d'une hiérarchie d'héritage

Fournit des informations sur la conception et l'implémentation d'une hiérarchie de classe.

Rubriques connexes

Interfaces en Visual Basic

Présente les interfaces et leur utilisation.

Programmation orientée objet dans Visual Basic

Introduit les concepts de la programmation orientée objet utilisés en Visual Basic.

Éléments fondamentaux de l'héritage

L'instruction **Inherits** permet de déclarer une nouvelle classe, appelée une *classe dérivée*, basée sur une classe existante, appelée une *classe de base*. Les classes dérivées héritent des propriétés, des méthodes, des événements, des champs et des constantes de la classe de base et peuvent les étendre. La section suivante décrit certaines règles de l'héritage et les modificateurs permettant de modifier la façon dont les classes peuvent hériter ou être héritées :

- Toutes les classes peuvent être héritées par défaut, sauf si elles sont marquées avec le mot clé **NotInheritable**. Les classes peuvent hériter d'autres classes dans votre projet ou de classes dans d'autres assemblys faisant référence à votre projet.
- Contrairement aux langages autorisant des héritages multiples, Visual Basic ne permet qu'un seul héritage dans des classes ; les classes dérivées ne peuvent avoir qu'une seule classe de base. Bien que les héritages multiples ne soient pas autorisés dans les classes, les classes peuvent implémenter plusieurs interfaces, qui peuvent accomplir efficacement les mêmes buts.
- Pour empêcher l'exposition d'éléments restreints dans une classe de base, le type d'accès d'une classe dérivée doit être égal à sa classe de base ou être plus restrictif. Par exemple, une classe **Public** ne peut pas hériter d'une classe **Friend** ou **Private**, et une classe **Friend** ne peut pas hériter d'une classe **Private**.

Modificateurs d'héritage

Visual Basic contient les instructions et les modificateurs de niveau classe suivants pour prendre en charge l'héritage :

- Instruction **Inherits** : spécifie la classe de base.
- Modificateur **NotInheritable** : empêche les programmeurs d'utiliser la classe comme une classe de base.
- Modificateur **MustInherit** : indique que la classe doit être utilisée uniquement comme une classe de base. Les instances de la classe **MustInherit** ne peuvent pas être créées directement ; elles ne peuvent être créées que comme des instances de classe de base d'une classe dérivée (d'autres langages de programmation, tels que C++ et C#, utilisent le terme *classe abstraite* pour décrire une telle classe).

Substitution de propriétés et de méthodes dans des classes dérivées

Par défaut, une classe dérivée hérite des propriétés et des méthodes de sa classe de base. Si une propriété ou une méthode héritée doit se comporter différemment dans la classe dérivée, elle peut être *substituée* : vous pouvez définir une nouvelle implémentation de la méthode dans la classe dérivée. Les modificateurs suivants sont utilisés pour contrôler la façon dont les propriétés et les méthodes sont substituées :

- **Overridable** : permet la substitution d'une propriété ou d'une méthode d'une classe dans une classe dérivée.
- **Overrides** : substitue une propriété ou une méthode **Overridable** définie dans la classe de base.
- **NotOverridable** : empêche la substitution d'une propriété ou d'une méthode dans une classe qui hérite. Les méthodes **Public** sont **NotOverridable** par défaut.
- **MustOverride** : la propriété ou la méthode doit être substituée par une classe dérivée. Lorsque le mot clé **MustOverride** est utilisé, la définition de méthode se compose uniquement de l'instruction **Sub**, **Function** ou **Property**. Aucune autre instruction n'est autorisée et, en particulier, il n'y a aucune instruction **End Sub** ni **End Function**. Les méthodes **MustOverride** doivent être déclarées dans des classes **MustInherit**.

Pour plus d'informations sur la substitution de méthodes, consultez Substitution de propriétés et de méthodes.

MyBase, mot clé

Le mot clé **MyBase** permet d'appeler des méthodes dans une classe de base lors de la substitution de méthodes dans une classe dérivée. Supposons par exemple que vous créez une classe dérivée qui substitue une méthode héritée d'une classe de base. La méthode substituée peut appeler la méthode dans la classe de base et modifier la valeur de retour, comme illustré dans le fragment de code suivant :

VB

```
Class DerivedClass
    Inherits BaseClass
```

```
    Public Overrides Function CalculateShipping( _
        ByVal Dist As Double, _
        ByVal Rate As Double) _
        As Double
        ' Call the method in the base class and modify the return value.
        Return MyBase.CalculateShipping(Dist, Rate) * 2
    End Function
End Class
```

La liste suivante décrit les restrictions de l'utilisation de **MyBase** :

- **MyBase** fait référence à la classe de base immédiate et à ses membres hérités. Il ne peut pas être utilisé pour accéder aux membres **Private** de cette classe.
- **MyBase** est un mot clé, et non un vrai objet. **MyBase** ne peut pas être assigné à une variable, passé à des procédures, ni utilisé dans une comparaison **Is**.
- La méthode qualifiée par **MyBase** ne doit pas être définie dans la classe de base immédiate, mais elle doit être définie dans une classe de base héritée indirectement. Pour qu'une référence qualifiée par **MyBase** soit correctement compilée, une classe de base doit contenir une méthode qui correspond au nom et aux types de paramètres qui apparaissent dans l'appel.
- Vous ne pouvez pas utiliser **MyBase** pour appeler des méthodes de la classe de base **MustOverride**.
- **MyBase** ne peut pas être utilisé pour se qualifier lui-même. Le code suivant est donc non valide :

  ```
  MyBase.MyBase.BtnOK_Click()
  ```

- **MyBase** ne peut pas être utilisé dans des modules.
- **MyBase** ne peut pas être utilisé pour accéder à des membres d'une classe de base marqués **Friend** si la classe de base se situe dans un assembly différent.

MyClass, mot clé

Le mot clé **MyClass** vous permet d'appeler une méthode **Overridable** implémentée dans votre classe et de garantir que l'implémentation de la méthode dans cette classe est appelée à la place d'une méthode substituée dans une classe dérivée.

- **MyClass** est un mot clé, et non un vrai objet. **MyClass** ne peut pas être assigné à une variable, passé à des procédures, ni utilisé dans une comparaison **Is**.
- **MyClass** fait référence à la classe conteneur et à ses membres hérités.
- **MyClass** peut être utilisé en tant que qualificateur pour les membres **Shared**.
- **MyClass** ne peut pas être utilisé dans des modules standard.
- **MyClass** peut être utilisé pour désigner une méthode définie dans une classe de base et ne possède aucune implémentation de la méthode fournie dans cette classe. La signification d'une telle référence est identique à **MyBase.***Method*.

Voir aussi

Tâches

Comment : créer des classes dérivées

Référence

Inherits, instruction

Concepts

Substitution de propriétés et de méthodes

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Comment : créer des classes dérivées

Avec l'instruction **Inherits**, une classe hérite de tous les membres non privés de la classe spécifiée.

Pour hériter d'une autre classe

- Ajoutez une instruction **Inherits** portant le nom d'une classe que vous souhaitez utiliser comme classe de base en tant que première instruction dans votre classe dérivée. L'instruction **Inherits** doit être la première instruction sans commentaire après l'instruction de classe.

Exemple

L'exemple suivant définit deux classes. La première classe est une classe de base comprenant deux méthodes. La seconde classe hérite des deux méthodes de la classe de base, substitue la seconde méthode et définit un champ intitulé `Field`.

VB

```
Class Class1
    Sub Method1()
        MsgBox("This is a method in the base class.")
    End Sub
    Overridable Sub Method2()
        MsgBox("This is another method in the base class.")
    End Sub
End Class

Class Class2
    Inherits Class1
    Public Field2 As Integer
    Overrides Sub Method2()
        MsgBox("This is a method in a derived class.")
    End Sub
End Class

Protected Sub TestInheritance()
    Dim C1 As New Class1
    Dim C2 As New Class2
    C1.Method1() ' Calls a method in the base class.
    C1.Method2() ' Calls another method from the base class.
    C2.Method1() ' Calls an inherited method from the base class.
    C2.Method2() ' Calls a method from the derived class.
End Sub
```

Lorsque vous exécutez la procédure `TestInheritance`, les messages suivants apparaissent :

```
This is a method in the base class.

This is another method in the base class.

This is a method in the base class.

This is a method in a derived class.
```

Voir aussi

Concepts

Substitution de propriétés et de méthodes

Modificateurs Override

Autres ressources

Héritage en Visual Basic

Propriétés, champs et méthodes de classe

Quand utiliser l'héritage

L'héritage est un concept de programmation utile, mais pouvant être utilisé de façon incorrecte. Les interfaces effectuent souvent mieux cette opération. Cette rubrique et la rubrique Quand utiliser des interfaces vous permettent de mieux déterminer les situations dans lesquelles il convient d'utiliser ces approches.

L'héritage est approprié dans les situations suivantes :

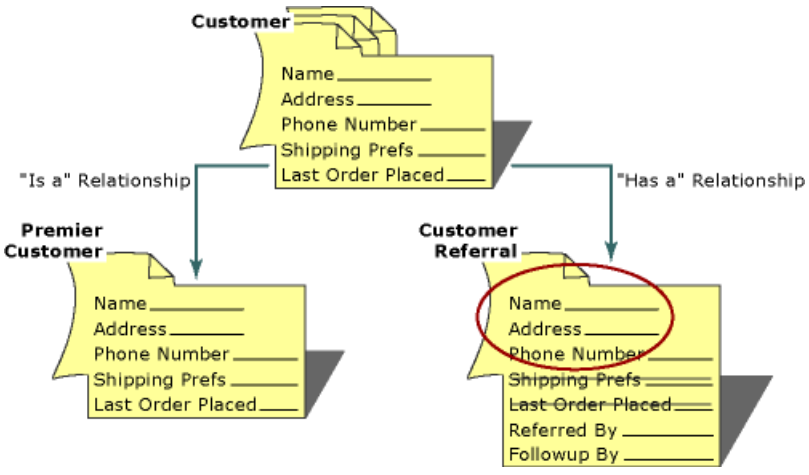
- Votre hiérarchie d'héritage est une relation de type « être » et non une relation de type « avoir ».
- Vous pouvez réutiliser du code des classes de base.
- Vous devez appliquer la même classe et les mêmes méthodes à divers types de données.
- La hiérarchie de classe est assez superficielle et il est peu probable que d'autres développeurs ajoutent des niveaux supplémentaires.
- Vous souhaitez apporter des modifications globales à des classes dérivées en modifiant une classe de base.

Ces différents aspects sont décrits, dans l'ordre, ci-dessous.

Héritage et relations de type « être »

Il existe deux façons de présenter les relations de classe dans la programmation orientée objet : les relations de type « être » et « avoir ». Dans une relation de type « être », la classe dérivée est un type de la classe de base. Par exemple, une classe intitulée `PremierCustomer` représente une relation de type « être » dont la classe de base est `Customer`, car un premier client est un client. Toutefois, une classe intitulée `CustomerReferral` représente une relation de type « avoir » avec la classe `Customer` car une référence de client appartient à un client, mais n'est pas un type de client.

Les objets d'une hiérarchie d'héritage doivent avoir une relation de type « être » avec leur classe de base car ils héritent des champs, des propriétés, des méthodes et des événements définis dans la classe de base. Les classes qui représentent une relation de type « avoir » avec d'autres classes ne conviennent pas aux hiérarchies d'héritage car elles peuvent hériter de propriétés et de méthodes inadaptées. Par exemple, si la classe `CustomerReferral` est dérivée de la classe `Customer` évoquée précédemment, elle peut hériter de propriétés absurdes, telles que `ShippingPrefs` et `LastOrderPlaced`. Une telle relation de type « avoir » doit être représentée en utilisant des classes ou des interfaces non apparentées. L'illustration suivante montre des exemples des relations « est un » et « a un ».



Classes de base et réutilisation de code

L'utilisation de l'héritage présente l'avantage de pouvoir réutiliser du code. Les classes bien conçues peuvent être déboguées une seule fois et réutilisées comme base des nouvelles classes.

Un exemple courant d'une réutilisation efficace du code concerne les bibliothèques qui gèrent des structures de données. Supposons par exemple que vous ayez une grande application d'entreprise qui gère plusieurs types de listes en mémoire. L'une de ces listes est une copie en mémoire de votre base de données des clients, lue depuis une base de données située au début de la session pour augmenter la vitesse. La structure de données peut se présenter comme suit :

VB

```
Class CustomerInfo
    Protected PreviousCustomer As CustomerInfo
```

```
    Protected NextCustomer As CustomerInfo
    Public ID As Integer
    Public FullName As String

    Public Sub InsertCustomer(ByVal FullName As String)
        ' Insert code to add a CustomerInfo item to the list.
    End Sub

    Public Sub DeleteCustomer()
        ' Insert code to remove a CustomerInfo item from the list.
    End Sub

    Public Function GetNextCustomer() As CustomerInfo
        ' Insert code to get the next CustomerInfo item from the list.
        Return NextCustomer
    End Function

    Public Function GetPrevCustomer() As CustomerInfo
        'Insert code to get the previous CustomerInfo item from the list.
        Return PreviousCustomer
    End Function
End Class
```

Votre application peut également posséder une liste similaire des produits ajoutés par un utilisateur à son panier d'achat, comme l'illustre le fragment de code suivant :

VB

```
Class ShoppingCartItem
    Protected PreviousItem As ShoppingCartItem
    Protected NextItem As ShoppingCartItem
    Public ProductCode As Integer
    Public Function GetNextItem() As ShoppingCartItem
        ' Insert code to get the next ShoppingCartItem from the list.
        Return NextItem
    End Function
End Class
```

En voici un modèle : deux listes se comportent de la même façon (insertions, suppressions et extractions), mais fonctionnent sur divers types de données. La gestion de deux bases de code pour effectuer essentiellement les mêmes fonctions n'est pas efficace. La solution la plus efficace consiste à isoler la gestion de la liste dans sa propre classe et d'hériter ensuite de cette classe pour divers types de données :

VB

```
Class ListItem
    Protected PreviousItem As ListItem
    Protected NextItem As ListItem
    Public Function GetNextItem() As ListItem
        ' Insert code to get the next item in the list.
        Return NextItem
    End Function
    Public Sub InsertNextItem()
        ' Insert code to add a item to the list.
    End Sub

    Public Sub DeleteNextItem()
        ' Insert code to remove a item from the list.
    End Sub

    Public Function GetPrevItem() As ListItem
        'Insert code to get the previous item from the list.
        Return PreviousItem
    End Function
End Class
```

La classe `ListItem` ne doit être déboguée qu'une seule fois. Vous pouvez ensuite construire des classes qui l'utilisent sans devoir vous préoccuper de la gestion de la liste. Par exemple :

VB

```
Class CustomerInfo
    Inherits ListItem
    Public ID As Integer
    Public FullName As String
End Class
Class ShoppingCartItem
    Inherits ListItem
    Public ProductCode As Integer
End Class
```

Bien que la réutilisation du code d'héritage soit un outil puissant, elle présente également certains risques. Même les systèmes les mieux conçus peuvent varier d'une façon dont les concepteurs n'avaient pas prévu. Les modifications d'une hiérarchie de classe existante peuvent parfois entraîner des conséquences inattendues. Pour plus d'informations, consultez la section « Problème de classe de base fragile » sous la rubrique Modifications du design de la classe de base après le déploiement.

Classes dérivées interchangeables

Les classes dérivées d'une hiérarchie de classe peuvent parfois être utilisées de façon interchangeable avec leur classe de base. Ce processus est appelé le polymorphisme d'héritage. Cette approche associe les meilleures fonctionnalités du polymorphisme d'interface avec l'option permettant de réutiliser ou de substituer du code d'une classe de base.

Un package de dessin représente un exemple où cette méthode pourrait s'avérer utile. Par exemple, examinons le fragment de code suivant qui n'utilise pas l'héritage :

VB

```
Sub Draw(ByVal Shape As DrawingShape, ByVal X As Integer, _
    ByVal Y As Integer, ByVal Size As Integer)

    Select Case Shape.type
        Case shpCircle
            ' Insert circle drawing code here.
        Case shpLine
            ' Insert line drawing code here.
    End Select
End Sub
```

Cette approche présente plusieurs problèmes. Si quelqu'un décide d'ajouter ultérieurement une option d'ellipse, le code source devra être modifié ; dans ce cas, il est possible que les utilisateurs cibles ne puissent plus accéder à votre code source. Un problème plus subtil est le suivant : pour dessiner une ellipse, vous avez besoin d'un autre paramètre (les ellipses ont un diamètre majeur et mineur) qui n'est pas adapté aux lignes. Si quelqu'un veut ensuite ajouter une polyligne (plusieurs lignes reliées), un autre paramètre doit être ajouté qui ne correspondrait pas aux autres cas.

L'héritage permet de résoudre la plupart de ces problèmes. Les classes de base bien conçues laissent l'implémentation de méthodes spécifiques aux classes dérivées, de sorte qu'il est possible d'accommoder tout type de forme. D'autres développeurs peuvent implémenter des méthodes dans des classes dérivées en utilisant la documentation de la classe de base. D'autres éléments de classe (tels que les coordonnées x et y) peuvent être construits dans la classe de base car tous les descendants les utilisent. Par exemple, `Draw` peut être une méthode **MustOverride** :

VB

```
MustInherit Class Shape
    Public X As Integer
    Public Y As Integer
    MustOverride Sub Draw()
End Class
```

Vous pouvez ensuite ajouter des paramètres à cette classe pour diverses formes. Par exemple, il est possible qu'une classe `Line` ne nécessite qu'un champ `Length` :

VB

```
Class Line
    Inherits Shape
    Public Length As Integer
    Overrides Sub Draw()
        ' Insert code here to implement Draw for this shape.
    End Sub
End Class
```

Cette approche est utile car les autres développeurs, qui ne peuvent pas accéder à votre code source, peuvent étendre leur classe de base avec de nouvelles classes dérivées si nécessaire. Par exemple, une classe intitulée `Rectangle` peut être dérivée de la classe `Line` :

VB

```
Class Rectangle
    Inherits Line
    Public Width As Integer
    Overrides Sub Draw()
        ' Insert code here to implement Draw for the Rectangle shape.
    End Sub
End Class
```

Cet exemple illustre la façon dont vous pouvez passer des classes générales à des classes très spécifiques en ajoutant des détails d'implémentation à chaque niveau.

À ce niveau, il est recommandé de réévaluer si la classe dérivée représente réellement une relation de type « être » ou s'il s'agit au contraire d'une relation de type « avoir ». Si la nouvelle classe rectangle se compose uniquement de lignes, l'héritage ne représente pas la méthode la mieux adaptée. Toutefois, si le nouveau rectangle est une ligne comprenant une propriété de largeur, la relation de type « être » est conservée.

Hiérarchies de classes superficielles

L'héritage est recommandé pour les hiérarchies de classes assez superficielles. Les hiérarchies de classe très profondes et complexes peuvent être difficiles à développer. La décision d'utiliser une hiérarchie de classe implique une analyse des avantages liés à l'utilisation d'une hiérarchie de classe par rapport à la complexité. En règle générale, vous devez limiter les hiérarchies à six niveaux maximum. Toutefois, la profondeur maximale d'une hiérarchie de classe particulière dépend d'un nombre de facteurs, y compris de la quantité de complexité à chaque niveau.

Modifications globales des classes dérivées par l'intermédiaire de la classe de base

L'une des fonctionnalités les plus puissantes de l'héritage est la possibilité d'apporter des modifications dans une classe de base pour les répercuter dans les classes dérivées. Si vous procédez avec prudence, il suffit de mettre à jour l'implémentation d'une seule méthode pour que des douzaines, voire des centaines de classes dérivées puissent utiliser le nouveau code. Toutefois, cette opération peut être dangereuse car de telles modifications peuvent engendrer des problèmes avec les classes héritées conçues par de tierces personnes. Vous devez être prudent pour garantir que la nouvelle classe de base est compatible avec les classes qui utilisent la classe d'origine. Vous devez particulièrement éviter de modifier le nom ou le type des membres de la classe de base.

Supposons par exemple que vous conceviez une classe de base avec un champ de type **Integer** pour stocker des informations concernant le code postal et que d'autres développeurs aient créé des classes dérivées qui utilisent le champ de code postal hérité. Supposons également que votre champ de code postal puisse stocker cinq chiffres, mais que le bureau de poste ait modifié les codes postaux en leur ajoutant un tiret et quatre chiffres supplémentaires. Dans le pire scénario, vous pouvez modifier le champ de la classe de base pour stocker une chaîne de 10 caractères, mais les autres développeurs doivent modifier et recompiler les classes dérivées afin d'utiliser la nouvelle taille et le nouveau type de données.

Le moyen le plus sûr pour modifier une classe de base est de lui ajouter de nouveaux membres. Par exemple, vous pouvez ajouter un nouveau champ pour stocker les quatre chiffres supplémentaires du code postal de l'exemple précédent. Les applications clientes peuvent ainsi être mises à jour pour utiliser le nouveau champ sans endommager les applications existantes. La possibilité d'étendre les classes de base dans une hiérarchie d'héritage est un avantage important qui n'existe

pas avec les interfaces.

Voir aussi

Concepts

Quand utiliser des interfaces

Modifications du design de la classe de base après le déploiement

Héritage et le .NET Framework

Les applications Visual Studio peuvent utiliser une grande partie des classes du .NET Framework comme classes de base. Les classes d'attributs, par exemple, sont toujours basées sur la classe Attribute. Parmi les autres classes de base utiles du .NET Framework figurent Component, à partir de laquelle on hérite pour définir des composants, et Exception, à partir de laquelle on hérite pour définir des classes d'exception personnalisées.

Le .NET Framework représente plus qu'une source pour les classes de base ; il garantit la compatibilité entre les classes créées avec des langages de programmation différents. Les classes Visual Basic peuvent hériter de classes dans des assemblys créés par tout langage pris en charge par Visual Studio. Les classes créées dans Visual Basic peuvent également être utilisées comme des classes de base par d'autres langages de programmation du .NET Framework.

Voir aussi

Tâches

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Concepts

Héritage et la classe Object de base

Quand utiliser l'héritage

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Vous pouvez dériver des classes Visual Basic 2005 à partir de classes **Public** dans des objets COM, même les classes créées avec des versions antérieures de Visual Basic. Les propriétés et les méthodes des classes héritées d'objets COM peuvent être substituées ou surchargées comme les propriétés et les méthodes de toute autre classe de base. L'héritage à partir des objets COM est utile lorsque vous ne voulez pas recompiler une bibliothèque de classes existante.

La procédure suivante explique comment utiliser Visual Basic 6.0 pour créer un objet COM contenant une classe, puis comment l'utiliser comme classe de base dans Visual Basic 2005.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour générer l'objet COM utilisé dans cette procédure pas à pas

1. Dans Visual Basic 6.0, ouvrez un nouveau projet de DLL ActiveX. Un projet nommé `Project1` est créé avec une classe nommée `Class1`.
2. Dans l'**Explorateur de projets**, cliquez avec le bouton droit sur **Project1**, puis cliquez sur **Propriétés**. La boîte de dialogue **Propriétés du projet** s'affiche.
3. Sous l'onglet **Général** de la boîte de dialogue **Propriétés du projet**, modifiez le nom du projet en tapant `ComObject1` dans le champ **Nom de projet**.
4. Dans l'**Explorateur de projets**, cliquez avec le bouton droit sur `Class1`, puis cliquez sur **Propriétés**. La fenêtre **Propriétés** de la classe s'affiche.
5. Affectez la valeur `MathFunctions` à la propriété **Name**.
6. Dans l'**Explorateur de projets**, cliquez avec le bouton droit sur `MathFunctions`, puis cliquez sur **Afficher le code**. L'**éditeur de code** s'affiche.
7. Ajoutez une variable locale pour contenir la valeur de propriété :

```
' Local variable to hold property value
Private mvarProp1 As Integer
```

8. Ajoutez les procédures de propriété Property `Let` et Property `Get` :

```
Public Property Let Prop1(ByVal vData As Integer)
   'Used when assigning a value to the property.
   mvarProp1 = vData
End Property
Public Property Get Prop1() As Integer
   'Used when retrieving a property's value.
   Prop1 = mvarProp1
End Property
```

9. Ajoutez une fonction :

```
Function AddNumbers( _
   ByVal SomeNumber As Integer, _
   ByVal AnotherNumber As Integer) _
   As Integer
```

```
    AddNumbers = SomeNumber + AnotherNumber
End Function
```

10. Créez et inscrivez l'objet OM en sélectionnant **Créer ComObject1.dll** dans le menu **Fichier**.

> **Remarque**
>
> Bien que vous puissiez également exposer une classe créée avec Visual Basic comme un objet COM, il ne s'agit pas d'un objet COM réel et il ne peut pas être utilisé dans cette procédure pas à pas. Pour plus d'informations, consultez Interopérabilité COM dans les applications .NET Framework.

Assemblys d'interopérabilité

Dans la procédure suivante, vous créez un assembly d'interopérabilité qui sert de pont entre le code non managé (tel qu'un objet COM) et le code managé que Visual Studio utilise. L'assembly d'interopérabilité créé par Visual Basic gère de nombreux détails de l'utilisation d'objets COM, par exemple le *marshaling d'interopérabilité*, le processus qui consiste à empaqueter des paramètres et des valeurs de retour dans des types de données équivalents lors de leurs déplacements vers et à partir des objets COM. La référence de l'application Visual Basic pointe vers l'assembly d'interopérabilité et non vers l'objet COM réel.

Pour utiliser un objet COM avec Visual Basic 2005

1. Ouvrez un nouveau projet d'application Windows Visual Basic.
2. Dans le menu **Projet**, cliquez sur **Ajouter une référence**.

   La boîte de dialogue **Ajouter une référence** s'affiche.
3. Dans l'onglet **COM**, double-cliquez sur `ComObject1` dans la liste **Nom du composant** et cliquez sur **OK**.
4. Dans le menu **Projet**, cliquez sur **Ajouter un nouvel élément**.

   La boîte de dialogue **Ajouter un nouvel élément** s'affiche.
5. Dans le volet **Modèles**, cliquez sur **Classe**.

   Le nom de fichier par défaut, `Class1.vb`, s'affiche dans le champ **Nom**. Remplacez ce champ par MathClass.vb et cliquez sur **Ouvrir**. Cela crée une classe nommée `MathClass` et affiche son code.
6. Ajoutez le code suivant en haut de `MathClass` pour hériter de la classe COM.

   VB

```
' The inherited class is called MathFunctions in the base class,
' but the interop assembly appends the word Class to the name.
Inherits ComObject1.MathFunctionsClass
```

7. Surchargez la méthode publique de la classe de base en ajoutant le code suivant à `MathClass` :

   VB

```
'  This method overloads the method AddNumbers from the base class.
Overloads Function AddNumbers( _
    ByVal SomeNumber As Integer, _
    ByVal AnotherNumber As Integer) _
    As Integer

    Return SomeNumber + AnotherNumber
End Function
```

8. Étendez la classe héritée en ajoutant le code suivant à `MathClass` :

VB

```
'  The following function extends the inherited class.
Function SubtractNumbers( _
    ByVal SomeNumber As Integer, _
    ByVal AnotherNumber As Integer) _
    As Integer

    Return AnotherNumber - SomeNumber
End Function
```

La nouvelle classe hérite des propriétés de la classe de base dans l'objet COM, surcharge une méthode et définit une nouvelle méthode pour étendre la classe.

Pour tester la classe héritée

1. Ajoutez un bouton à votre formulaire de départ, puis double-cliquez dessus pour afficher son code.
2. Dans la procédure de gestionnaire d'événements **Click** du bouton, ajoutez le code suivant pour créer une instance de `MathClass` et appeler les méthodes surchargées :

   VB

   ```
   Dim Result1 As Short
   Dim Result2 As Integer
   Dim Result3 As Integer
   Dim MathObject As New MathClass
   Result1 = MathObject.AddNumbers(4S, 2S) ' Add two Shorts.
   Result2 = MathObject.AddNumbers(4, 2) 'Add two Integers.
   Result3 = MathObject.SubtractNumbers(2, 4) ' Subtract 2 from 4.
   MathObject.Prop1 = 6 ' Set an inherited property.

   MsgBox("Calling the AddNumbers method in the base class " & _
          "using Short type numbers 4 and 2 = " & Result1)
   MsgBox("Calling the overloaded AddNumbers method using " & _
          "Integer type numbers 4 and 2 = " & Result2)
   MsgBox("Calling the SubtractNumbers method " & _
          "subtracting 2 from 4 = " & Result3)
   MsgBox("The value of the inherited property is " & _
           MathObject.Prop1)
   ```

3. Exécutez le projet en appuyant sur F5.

Lorsque vous cliquez sur le bouton du formulaire, la méthode `AddNumbers` est d'abord appelée avec des nombres de types de données **Short** et Visual Basic choisit la méthode appropriée à partir de la classe de base. Le second appel à `AddNumbers` est dirigé vers la méthode surchargée à partir de `MathClass`. Le troisième appel s'adresse à la méthode `SubtractNumbers`, qui étend la classe. La propriété de la classe de base est définie et la valeur s'affiche.

Étapes suivantes

Vous avez peut-être remarqué que la fonction surchargée `AddNumbers` semble posséder le même type de données que la méthode héritée à partir de la classe de base de l'objet COM. Ceci est dû au fait que les arguments et paramètres de la méthode de la classe de base sont définis comme des nombres entiers 16 bits dans Visual Basic 6.0, mais sont exposés comme des nombres entiers 16 bits de type **Short** dans Visual Basic 2005. La nouvelle fonction accepte les nombres entiers de 32 bits et surcharge la fonction de la classe de base.

Lorsque vous utilisez des objets COM, il est important de vérifier la taille et les types de données des paramètres. Par exemple, lorsque vous utilisez un objet COM qui accepte un objet de collection Visual Basic 6.0 en tant qu'argument, vous ne pouvez pas fournir de collection Visual Basic 2005. Pour plus d'informations sur les modifications du type de données, consultez Modifications du langage pour les utilisateurs de Visual Basic 6.0.

Les propriétés et les méthodes héritées des classes COM peuvent être substituées, ce qui signifie que vous pouvez déclarer une propriété ou une méthode locale qui remplace une propriété ou une méthode héritée d'une classe COM de base. Les règles de substitution des propriétés COM héritées sont similaires aux règles de substitution des autres propriétés et méthodes, à l'exception des cas suivants :

- si vous substituez une propriété ou une méthode héritée d'une classe COM, vous devez également remplacer toutes les autres propriétés et méthodes héritées ;
- Les propriétés qui utilisent des paramètres **ByRef** ne peuvent pas être substituées.

Voir aussi

Tâches

Comment : créer des classes dérivées

Référence

Inherits, instruction

Short, type de données (Visual Basic)

Concepts

Substitution de propriétés et de méthodes

Autres ressources

Interopérabilité COM dans les applications .NET Framework

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Héritage et la classe Object de base

Quasiment toutes vos actions dans Visual Basic reposent sur l'utilisation d'objets, y compris des entités qui ne sont pas forcément perçues comme des objets, par exemple les chaînes et les entiers. Tous les objets dans Visual Basic sont dérivés et héritent de méthodes d'une classe de base nommée System.Object. La classe **System.Object** fournit une classe de base commune pour tous les objets du .NET Framework, ce qui garantit l'interopérabilité des objets développés à l'aide de Visual Studio. Les nouvelles classes héritent implicitement de la classe **System.Object** ; il n'est pas nécessaire de nommer explicitement cette classe avec une instruction **Inherits**.

Parmi les méthodes dont héritent les objets de **System.Object**, l'une des plus utiles est la méthode GetType, qui permet de retourner le type exact de l'objet en cours.

Voir aussi

Référence

Object
GetType

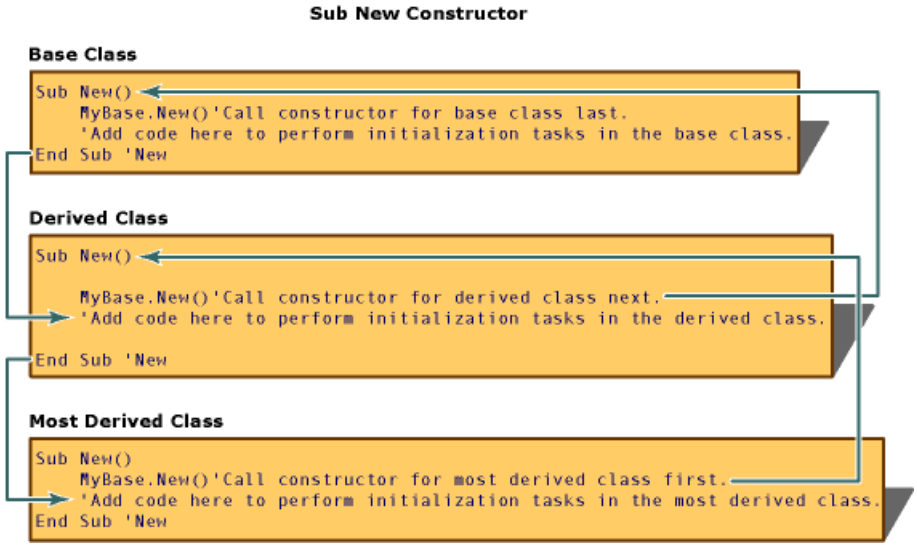
Concepts

Substitution de propriétés et de méthodes
Éléments fondamentaux de l'héritage

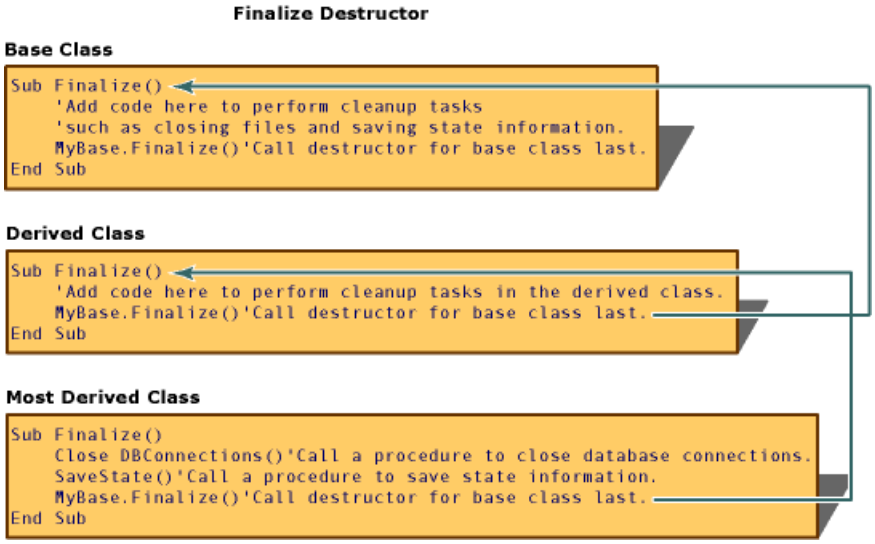
Fonctionnement des méthodes New et Finalize dans une hiérarchie de classe

À chaque fois qu'une instance d'une classe est créée, le common language runtime (CLR) tente d'exécuter une procédure nommée **New**, si elle existe dans cet objet. **New** est un type de procédure appelé un *constructor* qui est utilisé pour initialiser de nouveaux objets avant l'exécution d'un autre code dans un objet. Un constructeur **New** peut être utilisé pour ouvrir des fichiers, se connecter à des bases de données, initialiser des variables et réaliser d'autres tâches qui doivent être effectuées avant qu'un objet ne soit utilisé.

Lorsqu'une instance d'une classe dérivée est créée, le constructeur **Sub New** de la classe de base s'exécute premièrement, suivi des constructeurs des classes dérivées. En effet, la première ligne de code d'un constructeur **Sub New** utilise la syntaxe `MyBase.New()` pour appeler le constructeur de la classe situé immédiatement au-dessus de celle-ci dans la hiérarchie de classe. Le constructeur **Sub New** est ensuite appelé pour chaque classe dans la hiérarchie de classe jusqu'à ce que le constructeur de la classe de base soit atteint. À ce stade, le code du constructeur de la classe de base s'exécute, suivi du code de chacun des constructeurs de toutes les classes dérivées ; le code des classes plus dérivées s'exécute en dernier lieu.



Lorsqu'un objet n'est plus nécessaire, le CLR appelle la méthode Finalize pour cet objet avant de libérer sa mémoire. La méthode **Finalize** est appelée un *destructor* car elle effectue des tâches de nettoyage, telles qu'un enregistrement d'informations d'état, une fermeture de fichiers et des connexions à des bases de données, ainsi que d'autres tâches qui doivent être effectuées avant de libérer l'objet.



Voir aussi

Concepts

Durée de vie d'un objet : création et destruction des objets

Polymorphisme

Le *polymorphisme* permet de définir plusieurs classes pouvant être utilisées de manière interchangeable par le code client au moment de l'exécution, avec des méthodes et des propriétés différentes d'un point de vue fonctionnel, mais portant le même nom.

Dans cette section

Polymorphisme dans Visual Basic

Illustre diverses approches du polymorphisme.

Polymorphisme d'héritage

Présente l'utilisation de l'héritage pour le polymorphisme.

Polymorphisme d'interface

Présente l'utilisation des interfaces pour le polymorphisme.

Rubriques connexes

Quand utiliser des interfaces

Explique dans quels cas il convient d'utiliser des interfaces au lieu d'une hiérarchie d'héritage.

Quand utiliser l'héritage

Explique dans quels cas il convient d'utiliser l'héritage au lieu des interfaces.

Polymorphisme dans Visual Basic

À l'origine, les interfaces permettaient généralement d'accomplir le polymorphisme dans Visual Basic et elles peuvent toujours être utilisées dans ce but. À l'heure actuelle, Visual Basic permet toutefois d'utiliser l'héritage pour assurer également le polymorphisme.

Tout comme d'autres éléments de la programmation orientée objet, l'option à utiliser dépend de vos besoins spécifiques. En règle générale, utilisez l'héritage pour créer des fonctionnalités de base pouvant être étendues par les classes dérivées et utilisez les interfaces lorsque des fonctionnalités similaires doivent être fournies par plusieurs implémentations possédant peu de caractéristiques communes.

Voir aussi

Concepts

Polymorphisme d'héritage

Polymorphisme d'interface

Autres ressources

Design d'une hiérarchie d'héritage

Polymorphisme d'héritage

La majorité des systèmes de programmation orientée objet assurent le polymorphisme via l'héritage. Le polymorphisme d'héritage implique une définition de méthodes dans une classe de base et leur substitution par de nouvelles implémentations dans des classes dérivées.

Vous pouvez par exemple définir une classe `BaseTax` qui propose des fonctionnalités de base pour calculer le taux de T.V.A. dans un état. Les classes dérivées de `BaseTax`, telles que `CountyTax` ou `CityTax`, peuvent implémenter des méthodes, telles que `CalculateTax`, si nécessaire.

Le polymorphisme est issu de la possibilité d'appeler la méthode `CalculateTax` d'un objet appartenant à une classe dérivée de `BaseTax`, sans connaître la classe à laquelle appartient cet objet.

Dans l'exemple suivant, la procédure `TestPoly` illustre le polymorphisme d'héritage :

VB

```
' %5.3 State tax
Const StateRate As Double = 0.053
' %2.8 City tax
Const CityRate As Double = 0.028
Public Class BaseTax
    Overridable Function CalculateTax(ByVal Amount As Double) As Double
        ' Calculate state tax.
        Return Amount * StateRate
    End Function
End Class

Public Class CityTax
    ' This method calls a method in the base class
    ' and modifies the returned value.
    Inherits BaseTax
    Private BaseAmount As Double
    Overrides Function CalculateTax(ByVal Amount As Double) As Double
        ' Some cities apply a tax to the total cost of purchases,
        ' including other taxes.
        BaseAmount = MyBase.CalculateTax(Amount)
        Return CityRate * (BaseAmount + Amount) + BaseAmount
    End Function
End Class

Sub TestPoly()
    Dim Item1 As New BaseTax
    Dim Item2 As New CityTax
    ' $22.74 normal purchase.
    ShowTax(Item1, 22.74)
    ' $22.74 city purchase.
    ShowTax(Item2, 22.74)
End Sub

Sub ShowTax(ByVal Item As BaseTax, ByVal SaleAmount As Double)
    ' Item is declared as BaseTax, but you can
    ' pass an item of type CityTax instead.
    Dim TaxAmount As Double
    TaxAmount = Item.CalculateTax(SaleAmount)
    MsgBox("The tax is: " & Format(TaxAmount, "C"))
End Sub
```

Dans cet exemple, la procédure `ShowTax` accepte un paramètre intitulé `Item` de type `BaseTax`, mais vous pouvez également passer les classes dérivées de la classe `BaseTax`, comme `CityTax`. Cette conception présente l'avantage que vous pouvez ajouter de nouvelles classes dérivées de la classe `BaseTax` sans modifier le code client dans la procédure `ShowTax`.

Voir aussi

Concepts

Polymorphisme d'interface

Autres ressources

Design d'une hiérarchie d'héritage

Polymorphisme d'interface

Les interfaces vous permettent également d'accomplir le polymorphisme dans Visual Basic. Elles décrivent des propriétés et des méthodes, telles que des classes, mais contrairement aux classes, elles ne peuvent pas assurer d'implémentation. Les interfaces multiples permettent l'évolution des systèmes de composants logiciels sans endommager le code existant.

Pour accomplir le polymorphisme avec les interfaces, vous devez implémenter une interface de diverses manières dans plusieurs classes. Les applications clientes peuvent employer les implémentations nouvelles ou anciennes exactement de la même façon. L'avantage du polymorphisme d'interface est que vous n'avez pas besoin de recompiler les applications clientes existantes pour les utiliser avec les nouvelles implémentations d'interface.

L'exemple suivant définit une interface intitulée `Shape2` implémentée dans une classe nommée `RightTriangleClass2` et `RectangleClass2`. Une procédure nommée `ProcessShape2` appelle la méthode `CalculateArea` des instances de `RightTriangleClass2` ou `RectangleClass2` :

VB

```
Sub TestInterface()
    Dim RectangleObject2 As New RectangleClass2
    Dim RightTriangleObject2 As New RightTriangleClass2
    ProcessShape2(RightTriangleObject2, 3, 14)
    ProcessShape2(RectangleObject2, 3, 5)
End Sub

Sub ProcessShape2(ByVal Shape2 As Shape2, ByVal X As Double, _
      ByVal Y As Double)
    MsgBox("The area of the object is " _
       & Shape2.CalculateArea(X, Y))
End Sub

Public Interface Shape2
    Function CalculateArea(ByVal X As Double, ByVal Y As Double) As Double
End Interface

Public Class RightTriangleClass2
    Implements Shape2
    Function CalculateArea(ByVal X As Double, _
          ByVal Y As Double) As Double Implements Shape2.CalculateArea
        ' Calculate the area of a right triangle.
        Return 0.5 * (X * Y)
    End Function
End Class

Public Class RectangleClass2
    Implements Shape2
    Function CalculateArea(ByVal X As Double, _
          ByVal Y As Double) As Double Implements Shape2.CalculateArea
        ' Calculate the area of a rectangle.
        Return X * Y
    End Function
End Class
```

Voir aussi

Tâches

Comment : créer et implémenter des interfaces

Concepts

Polymorphisme d'héritage

Design d'une hiérarchie d'héritage

Il est plus facile d'implémenter des hiérarchies d'héritage que de les créer. Il est donc conseillé d'identifier clairement vos besoins avant de commencer le codage. Pour corriger des erreurs de design dans une hiérarchie de classe après l'implémentation, vous devrez peut-être modifier le code qui pourrait désactiver des applications existantes. Cette section décrit les considérations de design d'une hiérarchie d'héritage et vous procure des informations qui peuvent vous aider à éviter de telles erreurs.

Dans cette section

Considérations sur le design de hiérarchies de classes pour l'extensibilité

Présente le design de hiérarchies de classes de sorte qu'elles puissent être mises à jour ou étendues par d'autres développeurs.

Considérations sur la sélection de niveaux d'accès pour les méthodes

Illustre l'utilisation correcte des niveaux d'accessibilité dans des classes.

Modifications du design de la classe de base après le déploiement

Explique les problèmes auxquels sont confrontés les développeurs lorsqu'ils tentent d'apporter des modifications à une hiérarchie de classe.

Rubriques connexes

Quand utiliser des interfaces

Explique dans quels cas il convient d'utiliser des interfaces au lieu d'une hiérarchie d'héritage.

Interfaces en Visual Basic

Récapitule le design et l'implémentation d'interfaces.

Considérations sur le design de hiérarchies de classes pour l'extensibilité

Même les hiérarchies de classe bien conçues doivent évoluer au fil du temps. Les choix que vous effectuez lors de la conception d'une hiérarchie de classe peuvent simplifier votre travail ultérieurement.

Extension de hiérarchies de classe

La liste suivante contient quelques suggestions pour permettre d'étendre plus facilement vos hiérarchies de classe :

- Les hiérarchies de classe sont définies en partant des classes générales jusqu'aux classes spécifiques. Définissez des classes aussi générales que possible à chaque niveau d'une hiérarchie d'héritage. Les classes dérivées peuvent hériter, réutiliser et étendre des méthodes des classes de base.

  Supposons par exemple que vous conceviez une hiérarchie de classe qui décrit le matériel informatique. Lorsque vous commencez à décrire les périphériques de sortie, vous pouvez définir des classes appelées `Display`, `Printer` et `File`. Vous pouvez ensuite définir les classes qui implémentent les méthodes définies dans les classes de base. Par exemple, la classe `LCDDisplay` pourrait être dérivée de `Display` et implémenter une méthode nommée `EnterPowerSaveMode`.

- Définissez de grands types de données et de stockage pour éviter de devoir les modifier par la suite. Vous pouvez par exemple utiliser une variable de type **Long** même si vos données actuelles ne nécessitent qu'une variable standard de type **Integer**.

- Vous devez exposer uniquement les éléments qui sont nécessaires aux classes dérivées. Les champs et méthodes **Private** réduisent les conflits d'affectation de noms et empêchent les autres utilisateurs d'utiliser des éléments que vous devrez peut-être modifier ultérieurement.

- Les membres qui sont uniquement nécessaires aux classes dérivées devraient être marqués en tant que **Protected**. Cela garantit que seules les classes dérivées dépendent de ces membres et facilite la mise à jour de ces membres durant le développement.

- Assurez-vous que les méthodes de classe de base ne dépendent pas de membres **Overridable**, dont la fonctionnalité peut être modifiée par des classes qui héritent.

Voir aussi

Référence

MustInherit

MustOverride

Concepts

Considérations sur la sélection de niveaux d'accès pour les méthodes

Modifications du design de la classe de base après le déploiement

Considérations sur la sélection de niveaux d'accès pour les méthodes

Si vous appliquez les niveaux d'accès optimaux aux membres d'une hiérarchie de classe, vous pouvez plus facilement gérer la hiérarchie car vous pouvez contrôler la façon dont ces membres sont utilisés.

En règle générale, vous devez déclarer des membres de classe avec des modificateurs d'accès qui fournissent l'accès minimum. La restriction d'accès aux membres de classe réduit les conflits de noms et empêche une utilisation inattendue des méthodes.

Les membres de classe interne doivent être déclarés comme **Private** ; vous ne pouvez accéder à ces membres que depuis la classe dans laquelle ils ont été définis.

Les méthodes qui sont uniquement utilisées à l'intérieur d'une classe ou par des descendants d'une classe doivent utiliser le modificateur d'accès **Protected**. Un membre **Protected** est accessible à partir de la classe dans laquelle il est déclaré et à partir de toute classe dérivée de la classe ayant déclaré ce membre.

Les données membres **Friend** sont accessibles depuis l'extérieur d'une classe, mais uniquement par des modules faisant partie du projet lors de la définition de la classe.

Les données membres **Public** sont visibles à toute personne et sont souvent utilisés au bas d'une hiérarchie de classe.

Voir aussi

Concepts

Considérations sur le design de hiérarchies de classes pour l'extensibilité
Modifications du design de la classe de base après le déploiement

Modifications du design de la classe de base après le déploiement

Dans l'idéal, les hiérarchies de classe ne changent jamais après leur déploiement car des modifications minimes peuvent avoir des conséquences inattendues. En réalité, il n'est pas toujours possible d'éviter ces modifications : les exigences des produits peuvent varier et les spécifications de conception omettent parfois des éléments principaux. Une catégorie des problèmes liés à l'héritage est appelée *problème de classe de base fragile*.

Problème de classe de base fragile

Le principal inconvénient lié à l'utilisation des hiérarchies d'héritage est un problème connu sous le nom de *classes de base fragiles*. Les modifications apportées aux classes de base nécessitent souvent que la classe de base et que le code des classes dérivées soient modifiés, recompilés et redistribués. Ce problème est d'autant plus ardu lorsque plusieurs développeurs créent la classe de base et les classes dérivées, car chacune des parties peut refuser l'accès à son code source. Dans le pire scénario, un client peut utiliser par inadvertance la forme binaire compilée d'une classe de base mise à jour avec la version binaire d'origine incompatible des classes dérivées.

Les modifications de la classe de base pouvant endommager les classes dérivées sont notamment la substitution ou la modification des types de données des membres de la classe de base.

Limitation des problèmes de classe de base fragile

La meilleure solution pour éviter ce problème consiste à apporter des modifications uniquement dans les classes dérivées. Toutefois, cette solution n'est pas toujours possible car vous devez considérer chaque possibilité lors de la première version de la classe de base. Il n'est cependant pas toujours possible d'éviter des modifications imprévues de la classe de base, en dépit des efforts déployés.

Les classes **MustInherit** et les méthodes **MustOverride** permettent de réduire ce problème car les détails relatifs à l'implémentation sont déplacés vers les classes dérivées, et la nécessité de modifier la classe de base est réduite. Ce n'est toutefois pas toujours possible, même si vous disposez de la meilleure planification possible.

Les données membres masquées des classes dérivées sont également utiles, car ils réduisent les conflits de noms avec les membres de la classe de base.

Le moyen le plus sûr pour étendre les fonctionnalités d'une classe de base est de lui ajouter de nouveaux membres. La seule façon pour que de nouveaux membres interrompent une classe de base est lorsque la classe dérivée utilise le mot clé **Overloads** pour hériter de méthodes de la classe de base et introduire de nouvelles méthodes portant le même nom. Il est possible d'éviter ce problème en spécifiant explicitement dans la classe dérivée les méthodes de la classe de base qui doivent être héritées de la base, en redéfinissant ces méthodes et en les déléguant à celles-ci.

Toutes les classes de base sont fragiles à un certain point. Dans l'analyse finale, le problème de la classe de base fragile ne peut pas être évité ; vous pouvez toutefois le minimiser par une conception minutieuse des hiérarchies de classe pour réduire la nécessité de modifier la classe de base et par un test poussé permettant de déterminer le moment où ces modifications ne peuvent pas être évitées.

Voir aussi

Référence

Shadows

Autres ressources

Design d'une hiérarchie d'héritage

Membres partagés en Visual Basic

Les membres partagés sont des propriétés, des procédures et des champs qui sont partagés par toutes les instances d'une classe ou structure. Ils sont appelés membres statiques dans certains langages de programmation.

Champs et propriétés partagés

Les champs et propriétés partagés sont utiles lorsque certaines informations font partie d'une classe, mais ne sont pas spécifiques à une instance particulière d'une classe. Lorsque vous modifiez la valeur d'un champ et d'une propriété partagés, vous modifiez la valeur associée à la classe et à toutes les instances de la classe.

En revanche, modifier la valeur d'un champ ou d'une propriété non partagés associés à une instance particulière n'a aucune incidence sur la valeur de ce champ ou de cette propriété dans les autres instances de la classe. Des champs et propriétés non partagés existent indépendamment pour chaque instance d'une classe.

De cette façon, les champs et propriétés partagés se comportent comme des variables globales accessibles uniquement à partir d'instances d'une classe ou avec la qualification du nom de classe. Sans champs et propriétés partagés, vous devriez recourir à des variables au niveau du module pour obtenir le même résultat. Cependant, l'emploi de variables de module peut rendre les classes difficiles à comprendre et à gérer. De plus, utiliser ces variables de cette façon enfreint le concept d'encapsulation représenté par les classes.

Procédures partagées

Les procédures partagées sont des méthodes de classe qui ne sont pas associées à une instance spécifique d'une classe. Par exemple, la méthode Cos définie dans la classe Math est une méthode partagée. Il est possible d'appeler une procédure partagée en tant que méthode d'un objet ou directement à partir de la classe.

Les procédures et propriétés partagées n'ont pas accès aux instances de la classe. Pour cette raison, les méthodes partagées ne peuvent contenir que des références qualifiées à des données membres non partagées.

Remarque

Il est recommandé de ne pas écrire de code dans lequel vous accéderiez à un membre partagé par le biais d'une instance de l a classe. Ceci parce que le compilateur ignore la qualification du membre partagé et le traite comme si l'accès au membre pa rtagé avait été réalisé directement par le biais de la classe. Dans certaines situations, vous pouvez projeter que l'objet qualifia nt exécute du code et, par conséquent, que le compilateur Visual Basic génère un avertissement pour l'accès à un membre pa rtagé par le biais d'une instance d'une classe. IntelliSense n'affiche pas de membres partagés pour une instance de la classe.

Membres partagés - exemple

L'exemple suivant crée un champ partagé, deux champs d'instance et une méthode partagée pour illustrer le fonctionnement des membres partagés dans le code :

VB

```
Public Class Item
    Public Shared Count As Integer = 1
    Public Shared Sub ShareMethod()
        MsgBox("Current value of Count: " & Count)
    End Sub

    Public Sub New(ByVal Name As String)
        ' Use Count to initialize SerialNumber.
        Me.SerialNumber = Count
        Me.Name = Name
        ' Increment the shared variable
        Count += 1
    End Sub
    Public SerialNumber As Integer
    Public Name As String
    Public Sub InstanceMethod()
        MsgBox("Information in the first object: " & _
            Me.SerialNumber & vbTab & Me.Name)
    End Sub
End Class
```

```
Sub TestShared()
    ' Create two instances of the class.
    Dim part1 As New Item("keyboard")
    Dim part2 As New Item("monitor")

    part1.InstanceMethod()
    part2.InstanceMethod()
    Item.ShareMethod()
End Sub
```

Lorsque vous exécutez la procédure `TestShared`, deux instances de la classe sont créées. Le constructeur utilise le champ partagé `Count` pour initialiser le champ d'instance `SerialNumber`, puis incrémente `Count`. Cette technique donne automatiquement un numéro de série différent à chaque instance.

Après avoir créé les deux instances, la méthode d'instance `InstanceMethod` est appelée sur les deux objets et la méthode partagée `ShareMethod` est également appelée. Le résultat est le suivant :

```
Information in the first object: 1 keyboard

Information in the second object: 2 monitor

Current value of the shared Count field: 3
```

Voir aussi

Référence

Dim, instruction (Visual Basic)

Cos

Concepts

Structures et classes

Autres ressources

Propriétés, champs et méthodes de classe

Structures constituées à partir de vos propres types de données

Liaison anticipée et liaison tardive

Le compilateur Visual Basic exécute un processus appelé *binding* quand un objet est assigné à une variable objet. Un objet est à *liaison anticipée* (early binding) lorsqu'il est assigné à une variable déclarée comme étant d'un type d'objet spécifique. Les objets à liaison anticipée permettent au compilateur d'allouer de la mémoire et d'effectuer d'autres optimisations avant l'exécution d'une application. Par exemple, le fragment de code suivant déclare une variable comme étant du type FileStream :

VB

```
'  Create a variable to hold a new object.
Dim FS As System.IO.FileStream
' Assign a new object to the variable.
FS = New System.IO.FileStream("C:\tmp.txt", _
    System.IO.FileMode.Open)
```

FileStream étant un type d'objet spécifique, l'instance assignée à `FS` fait l'objet d'une liaison anticipée.

Par opposition, un objet est à *liaison tardive* (late binding) lorsque cet objet est assigné à une variable déclarée comme étant du type **Object**. Les objets de ce type peuvent contenir des références à n'importe quel objet, mais ne présentent pas nombre des avantages des objets à liaison anticipée. Par exemple, le fragment de code suivant déclare une variable objet qui doit contenir un objet retourné par la fonction **CreateObject** :

VB

```
' To use this example, you must have Microsoft Excel installed on your computer.
' Compile with Option Strict Off to allow late binding.
Sub TestLateBinding()
    Dim xlApp As Object
    Dim xlBook As Object
    Dim xlSheet As Object
    xlApp = CreateObject("Excel.Application")
    ' Late bind an instance of an Excel workbook.
    xlBook = xlApp.Workbooks.Add
    ' Late bind an instance of an Excel worksheet.
    xlSheet = xlBook.Worksheets(1)
    xlSheet.Activate()
    ' Show the application.
    xlSheet.Application.Visible = True
    ' Place some text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
End Sub
```

Avantages de la liaison anticipée

Vous devez utiliser des objets à liaison anticipée chaque fois que cela est possible, car ils permettent au compilateur d'effectuer d'importantes optimisations, avec pour résultat des applications plus efficaces. Ils sont beaucoup plus rapides que les objets à liaison tardive et facilitent la lecture et la gestion du code en indiquant avec exactitude les types d'objets utilisés. La liaison anticipée présente également l'avantage d'activer des fonctionnalités utiles, telles que la saisie semi-automatique du code et l'Aide dynamique, car l'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio peut déterminer avec précision le type d'objet utilisé lorsque vous modifiez le code. Elle réduit le nombre et la gravité des erreurs d'exécution en permettant au compilateur de signaler les erreurs lors de la compilation d'un programme.

Remarque

La liaison tardive peut seulement être utilisée pour accéder à des membres de type déclarés comme **Public**. L'accès à des me mbres déclarés comme **Friend** ou **Protected Friend** provoque une erreur d'exécution.

Voir aussi

Référence

Object, type de données
CreateObject, fonction (Visual Basic)

Concepts

Durée de vie d'un objet : création et destruction des objets

Obtention d'informations sur la classe au moment de l'exécution

Dans certaines situations, vous pouvez avoir besoin de travailler avec des variables définies avec le type **Object** plutôt qu'avec des variables définies avec des types d'objets plus spécifiques. Les variables définies comme **Object** peuvent contenir un objet de n'importe quel autre type de données (par exemple, Integer, Double ou Object).

Dans cette section

Détermination du type Object

Explique comment déterminer la classe à laquelle appartient un objet.

Appel d'une propriété ou méthode à l'aide d'un nom de chaîne

Décrit pas à pas le processus permettant de spécifier les arguments par leur nom plutôt que par leur position lorsque vous appelez une méthode.

Rubriques connexes

Objets en Visual Basic

Présente les concepts fondamentaux de la programmation orientée objet.

Détermination du type Object

Les variables objets génériques (c'est-à-dire les variables que vous déclarez comme étant de type **Object**) peuvent contenir les objets de n'importe quelle classe. Lorsque vous utilisez des variables de type **Object**, vous pouvez devoir prendre des mesures différentes en fonction de la classe de l'objet ; par exemple, certains objets peuvent ne pas prendre en charge une propriété ou méthode particulière. Visual Basic fournit deux moyens de déterminer le type d'objet stocké dans une variable objet : la fonction **TypeName** et l'opérateur **TypeOf...Is**.

TypeName et TypeOf...Is

La fonction **TypeName** retourne une chaîne et représente le meilleur choix lorsque vous devez stocker ou afficher le nom de classe d'un objet, comme illustré dans le fragment de code suivant :

VB

```
Dim Ctrl As Control = New TextBox
MsgBox(TypeName(Ctrl))
```

L'opérateur **TypeOf...Is** représente le meilleur choix lorsqu'il s'agit de tester le type d'un objet, car il est beaucoup plus rapide qu'une comparaison de chaîne équivalente à l'aide de la fonction **TypeName**. Le fragment de code suivant utilise **TypeOf...Is** dans une instruction **If...Then...Else**.

VB

```
If TypeOf Ctrl Is Button Then
    MsgBox("The control is a button.")
End If
```

Néanmoins, il convient d'apporter certaines précisions. L'opérateur **TypeOf...Is** retourne **True** si l'objet est d'un type spécifique ou s'il est dérivé d'un type spécifique. Quasiment toutes vos actions dans Visual Basic reposent sur l'utilisation d'objets, y compris des éléments qui ne sont pas forcément perçues comme des objets, par exemple les chaînes et les entiers. Ces objets sont dérivés et héritent des méthodes de Object. Lorsqu'un **Integer** lui est passé et qu'il est évalué avec **Object**, l'opérateur **TypeOf...Is** retourne **True**. L'exemple suivant indique que le paramètre `InParam` est à la fois du type **Object** et **Integer** :

VB

```
Sub CheckType(ByVal InParam As Object)
    ' Both If statements evaluate to True when an
    ' Integer is passed to this procedure.
    If TypeOf InParam Is Object Then
        MsgBox("InParam is an Object")
    End If
    If TypeOf InParam Is Integer Then
        MsgBox("InParam is an Integer")
    End If
End Sub
```

L'exemple suivant utilise à la fois **TypeOf...Is** et **TypeName** pour déterminer le type d'objet passé dans l'argument `Ctrl`. La procédure `TestObject` appelle `ShowType` à l'aide de trois sortes de contrôles différents.

Pour exécuter l'exemple

1. Créez un projet Application Windows et ajoutez un contrôle Button, un contrôle CheckBox et un contrôle RadioButton au formulaire.
2. À partir du bouton situé sur votre formulaire, appelez la procédure `TestObject`.
3. Ajoutez le code suivant à votre formulaire :

   VB

```vb
Sub ShowType(ByVal Ctrl As Object)
    'Use the TypeName function to display the class name as text.
    MsgBox(TypeName(Ctrl))
    'Use the TypeOf function to determine the object's type.
    If TypeOf Ctrl Is Button Then
        MsgBox("The control is a button.")
    ElseIf TypeOf Ctrl Is CheckBox Then
        MsgBox("The control is a check box.")
    Else
        MsgBox("The object is some other type of control.")
    End If
End Sub

Protected Sub TestObject()
    'Test the ShowType procedure with three kinds of objects.
    ShowType(Me.Button1)
    ShowType(Me.CheckBox1)
    ShowType(Me.RadioButton1)
End Sub
```

Voir aussi

Référence

Object, type de données
TypeName, fonction (Visual Basic)
If...Then...Else, instruction (Visual Basic)
String, type de données (Visual Basic)
Integer, type de données (Visual Basic)

Concepts

Appel d'une propriété ou méthode à l'aide d'un nom de chaîne

Autres ressources

Obtention d'informations sur la classe au moment de l'exécution

Appel d'une propriété ou méthode à l'aide d'un nom de chaîne

Dans la plupart des cas, vous pouvez découvrir les propriétés et méthodes d'un objet au moment du design et écrire du code pour les gérer. Toutefois, dans quelques cas vous ne connaîtrez peut-être pas les propriétés et méthodes d'un objet à l'avance ou vous voudrez tout simplement offrir à l'utilisateur la possibilité de spécifier les propriétés et d'exécuter les méthodes au moment de l'exécution.

CallByName, fonction

Prenons l'exemple d'une application cliente qui évalue des expressions entrées par l'utilisateur en passant un opérateur au composant COM. Supposons que vous voulez constamment ajouter de nouvelles fonctions au composant requérant de nouveaux opérateurs. Si vous utilisez les techniques d'accès standard, vous devrez recompiler et redistribuer l'application cliente avant qu'elle ne puisse utiliser les nouveaux opérateurs. Pour éviter ce problème, vous pouvez appeler la fonction **CallByName** pour passer les nouveaux opérateurs en tant que chaînes sans modifier l'application.

La fonction **CallByName** vous permet d'utiliser une chaîne pour spécifier une propriété ou une méthode au moment de l'exécution. La signature de la fonction **CallByName** ressemble à ceci :

Résultat = **CallByName**(*Objet*, *NomProcédure*, *TypeAppel*, *Arguments*())

Le premier argument, *Objet*, prend le nom de l'objet sur lequel vous voulez agir. L'argument *NomProcédure* accepte une chaîne contenant le nom de la procédure de méthode ou de propriété à appeler. L'argument *TypeAppel* accepte une constante représentant le type de procédure à appeler : une méthode (`Microsoft.VisualBasic.CallType.Method`), une lecture de propriété (`Microsoft.VisualBasic.CallType.Get`) ou une définition de propriété (`Microsoft.VisualBasic.CallType.Set`). L'argument *Arguments*, facultatif, accepte un tableau de type **Object** contenant n'importe quel argument de la procédure.

Vous pouvez utiliser la fonction **CallByName** avec des classes de votre solution en cours, mais elle est plus souvent employée pour accéder à des objets COM ou des objets d'assemblys .NET Framework.

Supposons que vous ajoutez une référence à un assembly contenant une classe appelée `MathClass` dotée d'une nouvelle fonction appelée `SquareRoot`, comme illustré dans le code suivant :

VB

```
Class MathClass
    Function SquareRoot(ByVal X As Double) As Double
        Return Math.Sqrt(X)
    End Function
    Function InverseSine(ByVal X As Double) As Double
        Return Math.Atan(X / Math.Sqrt(-X * X + 1))
    End Function
    Function Acos(ByVal X As Double) As Double
        Return Math.Atan(-X / Math.Sqrt(-X * X + 1)) + 2 * Math.Atan(1)
    End Function
End Class
```

Votre application peut utiliser des contrôles de zone de texte pour contrôler la méthode et les arguments qui seront appelés. Par exemple, si `TextBox1` contient l'expression à évaluer et si `TextBox2` est utilisé pour entrer le nom de la fonction, vous pouvez utiliser le code suivant pour appeler la fonction `SquareRoot` pour l'expression contenue dans `TextBox1` :

VB

```
Private Sub CallMath()
    Dim Math As New MathClass
    Me.TextBox1.Text = CStr(CallByName(Math, Me.TextBox2.Text, _
       Microsoft.VisualBasic.CallType.Method, TextBox1.Text))
End Sub
```

Si vous entrez "64" dans `TextBox1`, "SquareRoot" dans `TextBox2` et que vous appelez ensuite la procédure `CallMath`, la racine carrée du nombre contenu dans `TextBox1` est évaluée. Le code de cet exemple appelle la fonction `SquareRoot` (qui accepte une chaîne contenant l'expression à évaluer comme un argument obligatoire) et retourne "8" dans `TextBox1` (la racine carrée de 64). Évidemment, si l'utilisateur entre une chaîne non valide dans `TextBox2`, si la chaîne contient le nom d'une propriété et non celui d'une méthode ou si la méthode possède un argument obligatoire supplémentaire, il se produit une erreur

d'exécution. Vous devez ajouter un code robuste de gestion des erreurs lorsque vous utilisez la fonction **CallByName** afin d'anticiper les erreurs de ce type.

Remarque
CallByName peut être utile dans certaines situations, mais pesez bien ses avantages par rapport à ses inconvénients en matière de performances — l'appel d'une procédure à l'aide de cette fonction est un peu plus lent qu'un appel à liaison tardive. Si la fonction est appelée de façon répétitive, comme dans une boucle, **CallByName** peut avoir un impact négatif sur les performances.

Voir aussi

Tâches

Comment : exécuter plusieurs actions sur un objet

Référence

CallByName, fonction

Concepts

Détermination du type Object

Autres ressources

Obtention d'informations sur la classe au moment de l'exécution

COM Interop

Le modèle COM (Component Object Model) permet à un objet d'exposer ses fonctionnalités à d'autres composants et d'héberger des applications. La plupart des logiciels actuels comprennent des objets COM. Même si les assemblys .NET représentent le choix idéal pour les nouvelles applications, vous devez peut-être parfois utiliser des objets COM. Cette section aborde certains des problèmes associés à la création et à l'utilisation d'objets COM avec Visual Basic.

Dans cette section

Introduction à COM Interop

Présente une vue d'ensemble de l'interopérabilité COM.

Comment : référencer les objets COM à partir de Visual Basic

Décrit la procédure à suivre pour ajouter des références aux objets COM qui possèdent des bibliothèques de types.

Comment : utiliser les contrôles ActiveX

Illustre l'utilisation de contrôles ActiveX existants pour ajouter des fonctionnalités à la boîte à outils Visual Studio.

Procédure pas à pas : appel des API Windows

Vous guide tout au long du processus d'appel des API qui font partie du système d'exploitation Windows.

Comment : appeler des API Windows (Visual Basic)

Montre comment définir et appeler la fonction **MessageBox** dans User32.dll.

Procédure pas à pas : création d'objets COM avec Visual Basic 2005

Vous guide pas à pas dans le processus de création d'objets COM avec et sans le modèle de classe COM.

Dépannage des problèmes liés à l'interopérabilité

Aborde certains des problèmes qui peuvent se poser lors de l'utilisation de COM.

Interopérabilité COM dans les applications .NET Framework

Fournit une vue d'ensemble de la façon dont les objets COM et les objets .NET Framework doivent être utilisés dans la même application.

Rubriques connexes

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Illustre l'utilisation de classes dans les objets COM en tant que base de l'héritage en Visual Basic.

Interopération avec du code non managé

Décrit les services d'interopérabilité fournis par le Common Language Runtime.

Exposition de composants COM au .NET Framework

Décrit le processus d'appel de types COM via COM Interop.

Exposition de composants .NET Framework à COM

Décrit la préparation et l'utilisation de types managés à partir de COM.

Application d'attributs d'interopérabilité

Aborde les attributs que vous pouvez utiliser lors de l'utilisation du code non managé.

Introduction à COM Interop

Le modèle COM (Component Object Model) permet à un objet d'exposer ses fonctionnalités à d'autres composants et d'héberger des applications. Alors que les objets COM ont été essentiels à la programmation Windows pendant de nombreuses années, les applications conçues pour le CLR (Common Language Runtime) offrent de nombreux avantages.

Les applications .NET Framework remplaceront finalement celles qui ont été développées avec COM. Jusqu'alors, vous devrez peut-être utiliser ou créer les objets COM avec Visual Studio. L'interopérabilité avec COM, ou *COM Interop*, vous permet d'utiliser des objets COM existants lors de la transition vers le .NET Framework à votre propre rythme.

L'utilisation du .NET Framework pour créer des composants COM vous permet d'utiliser COM Interop sans inscription. Cela vous permet de contrôler quelle version de DLL est activée lorsque plusieurs versions sont installées sur un ordinateur et permet aux utilisateurs finals d'utiliser XCOPY ou FTP pour copier votre application vers un répertoire approprié sur leur ordinateur où elle peut être exécutée. Pour plus d'informations, consultez COM Interop sans inscription.

Données et code managés

Le code développé pour le .NET Framework est appelé *code managé* et contient des métadonnées utilisées par le CLR. Les données utilisées par les applications .NET Framework sont appelées *données managées* car le runtime gère les tâches relatives aux données, telles que l'allocation et la libération de la mémoire, ainsi que la vérification du type. Par défaut, Visual Basic 2005 utilise des données et du code managés, mais vous pouvez accéder aux données et code non managés des objets COM à l'aide des assemblys d'interopérabilité (décrits plus loin sur cette page).

Assemblys

Un assembly constitue le bloc de construction principal d'une application .NET Framework. Il s'agit d'un ensemble de fonctionnalités qui sont générées, dotées d'une version et déployées comme une seule unité d'implémentation contenant un ou plusieurs fichiers. Chaque assembly contient un manifeste d'assembly.

Bibliothèques de types et manifestes d'assembly

Les bibliothèques de types décrivent les caractéristiques des objets COM, telles que les noms des membres et les types de données. Les manifestes d'assembly exécutent la même fonction pour les applications .NET Framework. Ils contiennent des informations relatives aux éléments suivants :

- Identité d'assembly, version, culture et signature numérique
- Fichiers qui constituent l'implémentation de l'assembly
- Types et ressources qui composent l'assembly, y compris ceux qui en sont exportés
- Dépendances de compilation des autres assemblys
- Autorisations nécessaires au bon fonctionnement de l'assembly

Pour plus d'informations sur les assemblys et les manifestes d'assembly, consultez Assemblys.

Importation et exportation des bibliothèques de types

Visual Studio contient un utilitaire, Tlbimp, qui vous permet d'importer des informations à partir d'une bibliothèque de types vers une application .NET Framework. Vous pouvez générer des bibliothèques de types à partir des assemblys à l'aide de l'utilitaire Tlbexp.

Pour plus d'informations sur Tlbimp et Tlbexp, consultez Type Library Importer (Tlbimp.exe) et Type Library Exporter (Tlbexp.exe).

Assemblys d'interopérabilité

Les assemblys d'interopérabilité sont des assemblys .NET Framework qui servent de pont entre le code managé et non managé, en mappant les membres de l'objet COM à des membres managés du .NET Framework équivalents. Les assemblys d'interopérabilité créés par Visual Basic 2005 gèrent de nombreux détails de l'utilisation d'objets COM, par exemple le marshaling d'interopérabilité.

Marshaling d'interopérabilité

Toutes les applications .NET Framework partagent un ensemble de types courants qui permettent l'interopérabilité des objets, quel que soit le langage de programmation utilisé. Les paramètres et valeurs de retour des objets COM utilisent parfois des

types de données différents de ceux utilisés dans du code managé. Le *marshaling d'interopérabilité* est le processus qui consiste à empaqueter des paramètres et valeurs de retour dans des types de données équivalents lors de leurs déplacements vers et depuis des objets COM. Pour plus d'informations, consultez Vue d'ensemble du marshaling d'interopérabilité.

Voir aussi

Tâches

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Dépannage des problèmes liés à l'interopérabilité

Référence

Type Library Importer (Tlbimp.exe)

Type Library Exporter (Tlbexp.exe)

Concepts

Assemblys

Vue d'ensemble du marshaling d'interopérabilité

Autres ressources

COM Interop

Interopération avec du code non managé

COM Interop sans inscription

Comment : référencer les objets COM à partir de Visual Basic

Dans Visual Basic, l'ajout de références à des objets COM qui possèdent des bibliothèques de types se fait de la même manière que dans les versions antérieures. Toutefois, Visual Basic ajoute la création d'un assembly d'interopérabilité à la procédure. Les références aux membres de l'objet COM sont routées vers l'assembly d'interopérabilité, puis transmises à l'objet COM réel. Les réponses de l'objet COM sont routées vers l'assembly d'interopérabilité, puis transmises à votre application .NET Framework.

Pour ajouter des références aux objets COM

1. Dans le menu **Projet**, sélectionnez **Ajouter une référence**, puis cliquez sur l'onglet **COM** dans la boîte de dialogue.
2. Sélectionnez le composant à utiliser dans la liste des objets COM.
3. Pour simplifier l'accès à l'assembly d'interopérabilité, ajoutez une instruction **Imports** tout en haut de la classe ou du module où vous utiliserez l'objet COM.

   Visual Basic crée automatiquement des assemblys d'interopérabilité lorsque vous ajoutez une référence à une bibliothèque de types à partir de l'environnement de développement intégré (IDE, Integrated Development Environment). Lorsque vous travaillez à partir de la ligne de commande, vous pouvez employer l'utilitaire Tlbimp pour créer manuellement des assemblys d'interopérabilité.

Pour créer un assembly d'interopérabilité à l'aide de Tlbimp

1. Ajoutez l'emplacement de Tlbimp au chemin de recherche, s'il n'y figure pas déjà et que vous ne vous trouvez pas actuellement dans le répertoire où il est situé.
2. Appelez Tlbimp à partir d'une invite de commandes, en fournissant les informations suivantes :

   - Nom et emplacement de la DLL qui contient la bibliothèque de types
   - Nom et emplacement de l'espace de noms où les informations doivent être placées
   - Nom et emplacement de l'assembly d'interopérabilité cible

   Le code suivant en est un exemple :

   ```
   Tlbimp test3.dll /out:NameSpace1 /out:Interop1.dll
   ```

   Vous pouvez utiliser Tlbimp pour créer des assemblys d'interopérabilité pour les bibliothèques de types, voire pour des objets COM non inscrits. Toutefois, les objets COM désignés par le biais d'assemblys d'interopérabilité doivent être correctement inscrits sur l'ordinateur sur lequel ils doivent être utilisés. Vous pouvez inscrire un objet COM à l'aide de l'utilitaire Regsvr32 fourni avec le système d'exploitation Windows.

Voir aussi

Tâches

Procédure pas à pas : implémentation de l'héritage avec les objets COM
Dépannage des problèmes liés à l'interopérabilité

Référence

Type Library Importer (Tlbimp.exe)
Type Library Exporter (Tlbexp.exe)
Imports, instruction

Autres ressources

COM Interop

Comment : utiliser les contrôles ActiveX

Les contrôles ActiveX sont des composants ou objets COM que vous pouvez insérer dans une page Web ou une autre application pour réutiliser les fonctionnalités empaquetées programmées par un autre utilisateur. Vous pouvez utiliser des contrôles ActiveX développés pour Visual Basic 6.0 et les versions antérieures afin d'ajouter des fonctionnalités à la **Boîte à outils** de Visual Studio.

Pour ajouter des contrôles ActiveX à la boîte à outils

1. Dans le menu **Outils**, cliquez sur **Choisir des éléments de boîte à outils**.

   La boîte de dialogue **Choisir des éléments de boîte à outils** apparaît.

2. Cliquez sur l'onglet **Composants COM**.

3. Activez la case à cocher située en regard du contrôle ActiveX que vous voulez utiliser, puis cliquez sur **OK**.

   Le nouveau contrôle apparaît avec les autres outils de la **Boîte à outils**.

Remarque
Vous pouvez vous servir de l'utilitaire Aximp pour créer manuellement un assembly d'interopérabilité pour les contrôles ActiveX. Pour plus d'informations, consultez ActiveX Control Importer pour Windows Forms (Aximp.exe).

Voir aussi

Tâches

Comment : ajouter des contrôles ActiveX aux Windows Forms

Dépannage des problèmes liés à l'interopérabilité

Référence

Choisir des éléments de boîte à outils, boîte de dialogue (Visual Studio)

ActiveX Control Importer pour Windows Forms (Aximp.exe)

Concepts

Considérations sur l'hébergement d'un contrôle ActiveX dans un Windows Form

Autres ressources

COM Interop

Procédure pas à pas : appel des API Windows

Les API Windows sont des bibliothèques de liens dynamiques (DLL, Dynamic-Link Libraries) qui font partie intégrante du système d'exploitation Windows. Elles permettent d'effectuer des tâches lorsqu'il s'avère difficile d'écrire des procédures équivalentes. Par exemple, Windows propose une fonction nommée **FlashWindowEx** qui vous permet de varier l'aspect de la barre de titre d'une application entre des tons clairs et foncés.

L'avantage de l'utilisation d'API Windows dans votre code réside dans le gain de temps de développement, car elles contiennent des douzaines de fonctions utiles déjà écrites et prêtes à être utilisées. L'inconvénient des API Windows est qu'elles peuvent être complexes à utiliser et implacables lorsqu'une opération se déroule mal.

Les API Windows représentent une catégorie spéciale d'interopérabilité. Les API Windows n'utilisent pas de code managé, n'ont pas de bibliothèques de types intégrées et utilisent des types de données différents de ceux utilisés avec Visual Studio. En raison de ces différences, et comme les API Windows ne sont pas des objets COM, l'interopérabilité avec les API Windows et le .NET Framework est exécutée via un appel de plate-forme, ou PInvoke. L'appel de plate-forme est un service qui permet à du code managé d'appeler des fonctions non managées implémentées dans les DLL. Pour plus d'informations, consultez Consommation de fonctions DLL non managées. Vous pouvez employer PInvoke en Visual Basic en utilisant l'instruction **Declare** ou en appliquant l'attribut **DllImport** à une procédure vide.

Les appels API Windows représentaient auparavant une part importante de la programmation Visual Basic, mais sont rarement nécessaires avec Visual Basic 2005. Chaque fois que cela est possible, vous devez utiliser du code managé à partir du .NET Framework plutôt que les appels API Windows pour effectuer des tâches. Cette procédure pas à pas fournit des informations pour les cas dans lesquels l'utilisation des API Windows est nécessaire.

Remarque
Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Appels API via Declare

La façon la plus courante d'appeler les API Windows consiste à utiliser l'instruction **Declare**.

Pour déclarer une procédure DLL

1. Identifiez le nom de la fonction que vous voulez appeler, ses arguments, types d'arguments et valeur de retour, ainsi que le nom et l'emplacement de la DLL qui la contient.

Remarque
Pour plus d'informations sur les API Windows, consultez la documentation du kit Win32 SDK dans les API Windows du kit de développement Platform SDK. Pour plus d'informations sur les constantes utilisées par les API Windows, examin ez les fichiers d'en-tête, tels que Windows.h, fournis avec le Kit de développement Platform SDK.

2. Ouvrez un nouveau projet d'application Windows en cliquant sur **Nouveau** dans le menu **Fichier**, puis sur **Projet**. La boîte de dialogue **Nouveau projet** s'affiche.
3. Sélectionnez **Application Windows** dans la liste des modèles de projet Visual Basic. Le nouveau projet s'affiche.
4. Ajoutez la fonction **Declare** suivante à la classe ou au module dans lequel vous souhaitez utiliser la DLL :

   VB

   ```
   Declare Auto Function MBox Lib "user32.dll" Alias "MessageBox" ( _
       ByVal hWnd As Integer, _
       ByVal txt As String, _
       ByVal caption As String, _
       ByVal Typ As Integer) _
       As Integer
   ```

Parties de l'instruction Declare

L'instruction **Declare** comprend les éléments suivants.

Modificateur Auto

Le modificateur **Auto** indique au runtime de convertir la chaîne basée sur le nom de la méthode en fonction des règles du Common Language Runtime (ou du nom d'alias, s'il est indiqué).

Mots clés Lib et Alias

Le nom qui suit le mot clé **Function** est celui que votre programme utilise pour accéder à la fonction importée. Il peut s'agir du nom réel de la fonction que vous appelez, ou vous pouvez utiliser tout nom de procédure valide, puis le mot clé **Alias** pour indiquer le nom réel de cette fonction.

Spécifiez le mot clé **Lib** suivi du nom et de l'emplacement de la DLL qui contient la fonction que vous appelez. Vous n'avez pas besoin d'indiquer le chemin d'accès des fichiers situés dans les répertoires système Windows.

Utilisez le mot clé **Alias** si le nom de la fonction que vous appelez n'est pas un nom de procédure Visual Basic valide ou est en conflit avec le nom d'autres éléments de votre application. **Alias** indique le véritable nom de cette fonction.

Déclaration du type de données et des arguments

Déclarez les arguments et leurs types de données. Cette partie peut être complexe, car les types de données utilisés dans Windows ne correspondent pas aux types de données Visual Studio. Visual Basic effectue la plupart des tâches à votre place en convertissant des arguments en types de données compatibles, processus appelé *marshaling*. Vous pouvez contrôler de manière explicite la façon dont les arguments sont marshalés en utilisant l'attribut MarshalAsAttribute défini dans l'espace de noms System.Runtime.InteropServices.

Remarque
Les versions antérieures de Visual Basic vous ont permis de déclarer des paramètres **As Any**, ce qui signifie que les données de n'importe quel type de données pourraient être utilisées. Visual Basic suppose que vous utilisez un type de données spécifique pour toutes les instructions **Declare**.

Constantes API Windows

Certains arguments sont des combinaisons de constantes. Par exemple, l'API `MessageBox` illustrée dans cette procédure pas à pas accepte un argument de type nombre entier appelé `Typ` qui contrôle le mode d'affichage du message. Vous pouvez déterminer la valeur numérique de ces constantes par l'examen des instructions **#define** dans le fichier WinUser.h. Les valeurs numériques sont généralement affichées au format hexadécimal. Par conséquent, vous pouvez utiliser une calculatrice pour les ajouter et les convertir au format décimal. Par exemple, si vous voulez combiner les constantes pour le style exclamation `MB_ICONEXCLAMATION` 0x00000030 et le style Oui/Non `MB_YESNO` 0x00000004, vous pouvez ajouter les nombres et obtenir un résultat de 0x00000034, ou 52 décimales. Même si vous pouvez utiliser directement le résultat en décimales, il est conseillé de déclarer ces valeurs en tant que constantes dans votre application et de les combiner à l'aide de l'opérateur **Or**.

Pour déclarer des constantes pour les appels API Windows

1. Consultez la documentation pour la fonction Windows que vous appelez. Déterminez le nom des constantes utilisées et le nom du fichier .h qui contient les valeurs numériques de ces constantes.

2. Utilisez un éditeur de texte, tel que le Bloc-notes, pour afficher le contenu du fichier d'en-tête .h et rechercher les valeurs associées aux constantes que vous utilisez. Par exemple, l'API `MessageBox` utilise la constante `MB_ICONQUESTION` pour afficher un point d'interrogation dans le message. La définition de `MB_ICONQUESTION` se trouve dans WinUser.h et est définie comme suit :

   ```
   #define MB_ICONQUESTION 0x00000020L
   ```

3. Ajoutez des instructions **Const** équivalentes pour votre classe ou votre module pour rendre ces constantes disponibles pour l'application. Par exemple :

   VB

   ```
   Const MB_ICONQUESTION As Integer = &H20
   Const MB_YESNO As Integer = &H4
   Const IDYES As Integer = 6
   Const IDNO As Integer = 7
   ```

Pour appeler la procédure DLL

1. Ajoutez un bouton nommé `Button1` au formulaire de départ de votre projet, puis double-cliquez dessus pour afficher son code. Le gestionnaire d'événements du bouton s'affiche.
2. Ajoutez du code au gestionnaire d'événements `Click` pour le bouton que vous avez ajouté pour appeler la procédure et indiquez les arguments appropriés :

   VB

   ```
   Private Sub Button1_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles Button1.Click

       ' Stores the return value.
       Dim RetVal As Integer
       RetVal = MBox(0, "Declare DLL Test", "Windows API MessageBox", _
           MB_ICONQUESTION Or MB_YESNO)

       ' Check the return value.
       If RetVal = IDYES Then
           MsgBox("You chose Yes")
       Else
           MsgBox("You chose No")
       End If
   End Sub
   ```

3. Exécutez le projet en appuyant sur F5. Le message s'affiche avec les deux boutons de réponse **Oui** et **Non**. Cliquez sur l'un des deux.

Marshaling de données

Visual Basic convertit automatiquement les types de données des paramètres et valeurs de retour pour les appels API Windows, mais vous pouvez utiliser l'attribut **MarshalAs** pour indiquer de façon explicite les types de données non managés attendus par une API. Pour plus d'informations sur le marshaling d'interopérabilité, consultez Marshaling d'interopérabilité.

Pour utiliser Declare et MarshalAs dans un appel API

1. Déterminez le nom de la fonction à appeler, ses arguments, types de données et valeur de retour.
2. Pour simplifier l'accès à l'attribut **MarshalAs**, ajoutez une instruction **Imports** en haut du code de la classe ou du module, comme dans l'exemple suivant :

   VB

   ```
   Imports System.Runtime.InteropServices
   ```

3. Ajoutez un prototype de fonction pour la fonction importée à la classe ou au module que vous utilisez et appliquez l'attribut **MarshalAs** aux paramètres ou à la valeur de retour. Dans l'exemple suivant, un appel API qui attend le type **void*** est marshalé en tant que AsAny :

   VB

   ```
   Declare Sub SetData Lib "..\LIB\UnmgdLib.dll" ( _
       ByVal x As Short, _
       <MarshalAsAttribute(UnmanagedType.AsAny)> _
           ByVal o As Object)
   ```

Appels API via DllImport

L'attribut **DllImport** propose une seconde manière d'appeler des fonctions dans les DLL sans bibliothèque de types. **DllImport** équivaut approximativement à une instruction **Declare**, mais fournit plus de contrôle sur la manière d'appeler les fonctions.

Vous pouvez utiliser **DllImport** avec la plupart des appels API Windows tant que l'appel fait référence à une méthode partagée (parfois appelée *statique*). Vous ne pouvez pas utiliser des méthodes qui nécessitent une instance d'une classe. À la différence des instructions **Declare**, les appels **DllImport** ne peuvent pas utiliser l'attribut **MarshalAs**.

Pour appeler une API Windows à l'aide de l'attribut DllImport

1. Ouvrez un nouveau projet d'application Windows en cliquant sur **Nouveau** dans le menu **Fichier**, puis sur **Projet**. La boîte de dialogue **Nouveau projet** s'affiche.
2. Sélectionnez **Application Windows** dans la liste des modèles de projet Visual Basic. Le nouveau projet s'affiche.
3. Ajoutez un bouton nommé `Button2` au formulaire de départ.
4. Double-cliquez sur `Button2` pour ouvrir le mode code pour le formulaire.
5. Pour simplifier l'accès à **DllImport**, ajoutez une instruction **Imports** en haut du code de la classe du formulaire de départ :

   VB
   ```
   Imports System.Runtime.InteropServices
   ```

6. Déclarez une fonction vide qui précède l'instruction **End Class** pour le formulaire et affectez à la fonction le nom `MoveFile`.
7. Appliquez les modificateurs **Public** et **Shared** à la déclaration de fonction et définissez des paramètres pour `MoveFile` en fonction des arguments que la fonction API Windows utilise :

   VB
   ```
   Public Shared Function MoveFile( _
       ByVal src As String, _
       ByVal dst As String) _
       As Boolean
       ' Leave the body of the function empty.
   End Function
   ```

   Votre fonction peut porter n'importe quel nom de procédure valide ; l'attribut **DllImport** indique le nom dans la DLL. Il gère également le marshaling d'interopérabilité pour les paramètres et valeurs de retour ; vous pouvez donc choisir des types de données Visual Studio similaires à ceux que l'API utilise.
8. Appliquez l'attribut **DllImport** à la fonction vide. Le premier paramètre représente le nom et l'emplacement de la DLL qui contient la fonction que vous appelez. Vous n'avez pas besoin d'indiquer le chemin d'accès des fichiers situés dans les répertoires système Windows. Le second paramètre est un argument nommé qui indique le nom de la fonction dans API Windows : Dans cet exemple, l'attribut **DllImport** force les appels à `MoveFile` à être transférés à `MoveFileW` dans KERNEL32.DLL. La méthode `MoveFileW` copie un fichier à partir du chemin d'accès `src` vers le chemin d'accès `dst`.

   VB
   ```
   <DllImport("KERNEL32.DLL", EntryPoint:="MoveFileW", SetLastError:=True, _
       CharSet:=CharSet.Unicode, ExactSpelling:=True, _
       CallingConvention:=CallingConvention.StdCall)> _
   Public Shared Function MoveFile( _
       ByVal src As String, _
       ByVal dst As String) _
       As Boolean
       ' Leave the body of the function empty.
   End Function
   ```

9. Ajoutez le code au gestionnaire d'événements `Button2_Click` pour appeler la fonction :

   VB

```
Private Sub Button2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click

    Dim RetVal As Boolean = MoveFile("c:\tmp\Test.txt", "c:\Test.txt")
    If RetVal = True Then
        MsgBox("The file was moved successfully.")
    Else
        MsgBox("The file could not be moved.")
    End If
End Sub
```

10. Créez un fichier nommé Test.txt et placez-le dans le répertoire C:\Tmp de votre disque dur. Créez le répertoire Tmp si nécessaire.
11. Appuyez sur F5 pour démarrer l'application. Le formulaire principal s'affiche.
12. Cliquez sur **Button2**. Le message "The file was moved successfully" s'affiche si le fichier peut être déplacé.

Voir aussi

Référence

Declare, instruction
Auto
Alias
Lib
DllImportAttribute
MarshalAsAttribute

Concepts

Création de prototypes dans du code managé
Callback, exemple

Autres ressources

COM Interop

Comment : appeler des API Windows (Visual Basic)

Cet exemple définit et appelle la fonction **MessageBox** dans user32.dll et lui passe une chaîne.

Exemple

Cet exemple de code est également disponible sous forme d'extrait de code IntelliSense. Dans le sélecteur d'extrait de code, il se trouve en **Langage Visual Basic**. Pour plus d'informations, consultez Comment : insérer des extraits de code dans votre code (Visual Basic).

VB

```
' Defines the MessageBox function.
Public Class Win32
    Declare Auto Function MessageBox Lib "user32.dll" ( _
        ByVal hWnd As Integer, ByVal txt As String, _
        ByVal caption As String, ByVal Type As Integer) _
        As Integer
End Class

' Calls the MessageBox function.
Public Class DemoMessageBox
    Public Shared Sub Main()
        Win32.MessageBox(0, "Here's a MessageBox", "Platform Invoke Sample", 0)
    End Sub
End Class
```

Compilation du code

Cet exemple nécessite :

- Référence à l'espace de noms System.

Programmation fiable

Les conditions ci-dessous peuvent générer une exception.

- La méthode n'est pas statique, elle est abstraite ou a été définie précédemment. Le type parent est une interface, ou la longueur de *name* ou *dllName* est zéro. (ArgumentException)
- Le *name* ou *dllName* est **Nothing**. (ArgumentNullException)
- Le type conteneur a été créé précédemment à l'aide de `CreateType`. (InvalidOperationException)

Voir aussi

Tâches

Procédure pas à pas : appel des API Windows

Concepts

Présentation détaillée de l'appel de plate-forme
Exemples d'appel de plate-forme
Consommation de fonctions DLL non managées
Définition d'une méthode avec l'émission de réflexion

Autres ressources

COM Interop

Procédure pas à pas : création d'objets COM avec Visual Basic 2005

Lors de la création de nouvelles applications ou de nouveaux composants, il est conseillé de créer des assemblys .NET Framework. Toutefois, Visual Basic 2005 facilite également l'exposition d'un composant .NET Framework à COM. Cela vous permet de fournir de nouveaux composants pour des suites d'applications héritées (legacy) qui requièrent des composants COM. Cette procédure pas à pas montre comment utiliser Visual Basic pour exposer des objets .NET Framework en tant qu'objets COM, avec et sans le modèle de classe COM.

La façon la plus simple d'exposer des objets COM consiste à utiliser le modèle de classe COM. Ce modèle crée une classe, puis configure votre projet de façon à générer la couche classe et interopérabilité en tant qu'objet COM et l'inscrire auprès du système d'exploitation.

Remarque

Bien que vous puissiez également exposer une classe créée avec Visual Basic en tant qu'objet COM pour le code non managé à utiliser, il ne s'agit pas d'un véritable objet COM et il ne peut pas être utilisé par Visual Basic. Pour plus d'informations, consultez Interopérabilité COM dans les applications .NET Framework.

Remarque

Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent différer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de paramètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour créer un objet COM à l'aide du modèle de classe COM

1. Ouvrez un nouveau projet d'application Windows à partir du menu **Fichier** en cliquant sur **Nouveau projet**.
2. Dans la boîte de dialogue **Nouveau projet** sous le champ **Types de projets**, assurez-vous que Windows est sélectionné. Sélectionnez **Bibliothèque de classes** dans la liste **Modèles**, puis cliquez sur **OK**. Le nouveau projet s'affiche.
3. Sélectionnez **Ajouter un nouvel élément** dans le menu **Projet**. La boîte de dialogue **Ajouter un nouvel élément** s'affiche.
4. Sélectionnez **Classe COM** dans la liste **Modèles**, puis cliquez sur **Ajouter**. Visual Basic ajoute une nouvelle classe et configure le nouveau projet pour COM Interop.
5. Ajoutez du code, par exemple des propriétés, méthodes et événements, à la classe COM.
6. Sélectionnez **Générer ClassLibrary1** dans le menu **Générer**. Visual Basic génère l'assembly et inscrit l'objet COM auprès du système d'exploitation.

Création d'objets COM sans le modèle de classe COM

Vous pouvez également créer une classe COM manuellement au lieu d'utiliser le modèle de classe COM. Cette procédure est utile lorsque vous travaillez à partir de la ligne de commande ou lorsque vous souhaitez davantage de contrôle sur le mode de définition des objets COM.

Pour configurer votre projet pour générer un objet COM

1. Ouvrez un nouveau projet d'application Windows à partir du menu **Fichier** en cliquant sur **Nouveau Projet**.
2. Dans la boîte de dialogue **Nouveau projet** sous le champ **Types de projets**, assurez-vous que Windows est sélectionné. Sélectionnez **Bibliothèque de classes** dans la liste **Modèles**, puis cliquez sur **OK**. Le nouveau projet s'affiche.
3. Dans l'**Explorateur de solutions**, cliquez avec le bouton droit sur le projet, puis cliquez sur **Propriétés**. Le **Concepteur de projets** s'affiche.
4. Cliquez sur l'onglet **Compiler**.
5. Activez la case à cocher **Inscrire pour COM Interop**.

Pour configurer le code de votre classe pour créer un objet COM

1. Dans l'**Explorateur de solutions**, double-cliquez sur **Class1.vb** pour afficher son code.
2. Renommez la classe `ComClass1`.
3. Ajoutez les constantes suivantes à `ComClass1`. Elles stockeront les constantes GUID (identificateur global unique) dont les objets COM doivent être assortis.

   VB
   ```
   Public Const ClassId As String = ""
   Public Const InterfaceId As String = ""
   Public Const EventsId As String = ""
   ```
4. Pour obtenir un GUID, lancez Guidgen.exe pour démarrer l'utilitaire Guidgen. Sélectionnez **Registry Format** dans la liste des formats fournie par l'application Guidgen. Cliquez sur le bouton **Nouveau GUID** pour générer le GUID et cliquez sur le bouton **Copier** pour copier le GUID dans le Presse-papiers.
5. Remplacez la chaîne vide de `ClassId` par le GUID et supprimez les accolades de début et de fin. Par exemple, si le GUID fourni par Guidgen est `"{2C8B0AEE-02C9-486e-B809-C780A11530FE}"`, votre code doit s'afficher comme suit :

   VB
   ```
   Public Const ClassId As String = "2C8B0AEE-02C9-486e-B809-C780A11530FE"
   ```
6. Répétez les étapes précédentes pour les constantes `InterfaceId` et `EventsId`, comme dans l'exemple suivant.

   VB
   ```
   Public Const InterfaceId As String = "3D8B5BA4-FB8C-5ff8-8468-11BF6BD5CF91"
   Public Const EventsId As String = "2B691787-6ED7-401e-90A4-B3B9C0360E31"
   ```

   Remarque

   Assurez-vous que les GUID sont nouveaux et uniques, faute de quoi votre composant COM pourrait présenter des conf lits avec d'autres composants COM.
7. Ajoutez l'attribut **ComClass** à `ComClass1`, en indiquant les GUID de l'ID de la classe, de l'interface et des événements, comme dans l'exemple suivant :

   VB
   ```
   <ComClass(ComClass1.ClassId, ComClass1.InterfaceId, ComClass1.EventsId)> _
   Public Class ComClass1
   ```
8. Les classes COM doivent avoir un constructeur `Public Sub New()` sans paramètre, sans quoi elles ne seront pas inscrites correctement. Ajoutez un constructeur sans paramètre à la classe :

   VB
   ```
   Public Sub New()
       MyBase.New()
   End Sub
   ```
9. Ajoutez des propriétés, méthodes et événements à la classe, en la terminant par une instruction **End Class**. Sélectionnez **Générer la solution** dans le menu **Générer**. Visual Basic génère l'assembly et inscrit l'objet COM auprès du système

d'exploitation.

> **Remarque**
>
> Les objets COM que vous générez avec Visual Basic 2005 ne peuvent pas être utilisés par d'autres applications Visual Basic 2005, car ils ne constituent pas de véritables objets COM. Les tentatives d'ajout de références à ces objets COM généreront une erreur. Pour plus d'informations, consultez Interopérabilité COM dans les applications .NET Framework.

Voir aussi

Tâches

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Dépannage des problèmes liés à l'interopérabilité

Référence

ComClassAttribute, classe

#Region, directive

Autres ressources

COM Interop

Interopérabilité COM dans les applications .NET Framework

Dépannage des problèmes liés à l'interopérabilité

Lors d'interopérations entre COM et le code managé du .NET Framework, un ou plusieurs des problèmes courants suivants peuvent se poser.

Marshaling d'interopérabilité

Il se peut que vous deviez parfois utiliser des types de données qui ne font pas partie du .NET Framework. Les assemblys d'interopérabilité gèrent la plupart du travail pour les objets COM, mais vous devez peut-être contrôler les types de données utilisés lorsque des objets managés sont exposés à COM. Par exemple, les structures des bibliothèques de classes doivent indiquer le type non managé BStr sur les chaînes envoyées à des objets COM créés par Visual Basic 6.0 et par des versions antérieures. Dans ce cas, vous pouvez utiliser l'attribut MarshalAsAttribute pour que les types managés soient exposés en tant que types non managés.

Exportation de chaînes de longueur fixe dans du code non managé

Dans Visual Basic 6.0 et les versions antérieures, les chaînes sont exportées dans les objets COM sous la forme de séquences d'octets sans caractère null de fin. Dans un souci de compatibilité avec d'autres langages, Visual Basic 2005 inclut un caractère de fin lors de l'exportation de chaînes. La meilleure façon de résoudre cette incompatibilité consiste à exporter les chaînes qui n'ont pas ce caractère comme des tableaux de **Byte** ou **Char**.

Exportation de hiérarchies d'héritage

Les hiérarchies de classes managées s'aplatissent lorsqu'elles sont exposées sous forme d'objets COM. Par exemple, si vous définissez une classe de base avec un membre, puis héritez de cette classe dans une classe dérivée qui est exposée sous forme d'objet COM, les clients qui utilisent la classe dérivée dans l'objet COM ne sont pas en mesure d'utiliser les membres hérités. Les membres de la classe de base ne sont accessibles à partir d'objets COM que sous forme d'instances d'une classe de base, et uniquement si la classe de base est également créée sous forme d'objet COM.

Méthodes surchargées

Bien que vous puissiez créer des méthodes surchargées avec Visual Basic, elles ne sont pas prises en charge par COM. Lorsqu'une classe qui contient des méthodes surchargées est exposée comme objet COM, de nouveaux noms de méthodes sont générés pour les méthodes surchargées.

Par exemple, considérez une classe avec deux surcharges de la méthode `Synch`. Lorsque la classe est exposée comme objet COM, les nouveaux noms de méthodes générés peuvent être `Synch` et `Synch_2`.

Le changement de nom peut provoquer deux problèmes pour les consommateurs de l'objet COM.

1. Il se peut que les clients n'attendent pas les noms de méthodes générés.
2. Les noms de méthodes générés dans la classe exposée comme objet COM peuvent changer lorsque de nouvelles surcharges sont ajoutées à la classe ou sa classe de base, ce qui peut provoquer des problèmes de versioning.

Pour résoudre les deux problèmes, donnez à chaque méthode un nom unique au lieu d'utiliser la surcharge, lors du développement d'objets qui seront exposés comme objets COM.

Utilisation d'objets COM via les assemblys d'interopérabilité

Vous utilisez presque les assemblys d'interopérabilité comme s'il s'agissait de remplacements de code managé pour les objets COM qu'ils représentent. Toutefois, dans la mesure où ce sont des wrappers et non des objets COM réels, il existe quelques différences entre l'utilisation d'assemblys d'interopérabilité et celle d'assemblys standard. Ces différences comprennent l'exposition des classes ainsi que les types de données pour les paramètres et les valeurs de retour.

Classes exposées à la fois sous forme d'interfaces et de classes

À la différence des classes dans les assemblys standard, les classes COM sont exposées dans les assemblys d'interopérabilité à la fois sous la forme d'une interface et d'une classe qui représente la classe COM. Le nom de l'interface est identique à celui de la classe COM. Le nom de la classe d'interopérabilité est le même que celui de la classe COM d'origine, auquel le mot "Class" a été ajouté. Par exemple, supposons que vous avez un projet avec une référence à un assembly d'interopérabilité pour un objet COM. Si le nom de la classe COM est `MyComClass`, IntelliSense et l'Explorateur d'objets affichent une interface nommée `MyComClass` et une classe nommée `MyComClassClass`.

Création d'instances d'une classe .NET Framework

En règle générale, vous créez une instance d'une classe .NET Framework en utilisant l'instruction **New** avec un nom de classe.

Le fait d'avoir une classe COM représentée par un assembly d'interopérabilité est le seul cas dans lequel vous pouvez utiliser l'instruction **New** avec une interface. À moins d'utiliser la classe COM avec une instruction **Inherits**, vous pouvez utiliser l'interface de la même manière qu'une classe. Le code suivant montre comment créer un objet `Command` dans un projet qui fait référence à l'objet COM de la bibliothèque Microsoft ActiveX Data Objects 2.8 :

VB

```
Dim cmd As New ADODB.Command
```

Toutefois, si vous utilisez la classe COM comme base d'une classe dérivée, vous devez utiliser la classe d'interopérabilité qui représente la classe COM, comme dans le code suivant :

VB

```
Class DerivedCommand
    Inherits ADODB.CommandClass
End Class
```

Remarque

Les assemblys d'interopérabilité implémentent de façon implicite les interfaces qui représentent des classes COM. Vous ne devez pas essayer d'utiliser l'instruction **Implements** pour implémenter ces interfaces afin de ne pas provoquer d'erreur.

Types de données pour les paramètres et valeurs de retour

À la différence des membres des assemblys standard, ceux des assemblys d'interopérabilité peuvent posséder des types de données différents de ceux utilisés dans la déclaration initiale de l'objet. Même si les assemblys d'interopérabilité convertissent de façon implicite les types COM en types Common Language Runtime compatibles, vous devez prêter attention aux types de données utilisés des deux côtés pour éviter les erreurs d'exécution. Par exemple, dans les objets COM créés avec Visual Basic 6.0 et les versions antérieures, les valeurs du type **Integer** adoptent le type .NET Framework équivalent, **Short**. Il est recommandé d'utiliser l'Explorateur d'objets pour examiner les caractéristiques des membres importés avant de les utiliser.

Méthodes COM de niveau module

La plupart des objets COM sont utilisés par la création d'une instance de classe COM avec le mot clé **New**, puis par l'appel aux méthodes associées à ces objets. Une exception à cette règle concerne les objets COM qui contiennent des classes COM **AppObj** ou **GlobalMultiUse**. Ces classes sont semblables aux méthodes de niveau de module dans les classes Visual Basic 2005. Visual Basic 6.0 et les versions antérieures créent implicitement des instances de ces objets pour vous la première fois que vous appelez l'une de leurs méthodes. Par exemple, dans Visual Basic 6.0, vous pouvez ajouter une référence à la bibliothèque d'objets Microsoft DAO 3.6 et appeler la méthode `DBEngine` sans créer d'instance au préalable :

```
Dim db As DAO.Database
' Open the database.
Set db = DBEngine.OpenDatabase("C:\nwind.mdb")
' Use the database object.
```

Visual Basic 2005 exige que vous créiez toujours des instances d'objets COM pour pouvoir utiliser leurs méthodes. Pour employer ces méthodes dans Visual Basic 2005, déclarez une variable de la classe voulue et assignez l'objet à la variable objet à l'aide du nouveau mot clé. Vous pouvez utiliser le mot clé **Shared** pour vous assurer qu'une seule instance de la classe est créée.

VB

```
' Class level variable.
Shared DBEngine As New DAO.DBEngine

Sub DAOOpenRecordset()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim fld As DAO.Field
    ' Open the database.
    db = DBEngine.OpenDatabase("C:\nwind.mdb")

    ' Open the Recordset.
```

```
    rst = db.OpenRecordset( _
        "SELECT * FROM Customers WHERE Region = 'WA'", _
        DAO.RecordsetTypeEnum.dbOpenForwardOnly, _
        DAO.RecordsetOptionEnum.dbReadOnly)
    ' Print the values for the fields in the debug window.
    For Each fld In rst.Fields
        Debug.WriteLine(fld.Value.ToString & ";")
    Next
    Debug.WriteLine("")
    ' Close the Recordset.
    rst.Close()
End Sub
```

Erreurs non gérées dans les gestionnaires d'événements

L'un des problèmes courants d'interopérabilité concerne les erreurs dans les gestionnaires d'événements qui gèrent les événements déclenchés par des objets COM. Ces erreurs sont ignorées, sauf si vous les recherchez spécifiquement à l'aide d'instructions **On Error** ou **Try...Catch...Finally**. L'exemple suivant est extrait d'un projet Visual Basic 2005 qui fait référence à l'objet COM de la bibliothèque Microsoft ActiveX Data Objects 2.8.

VB

```
' To use this example, add a reference to the
'     Microsoft ActiveX Data Objects 2.8 Library
' from the COM tab of the project references page.
Dim WithEvents cn As New ADODB.Connection
Sub ADODBConnect()
    cn.ConnectionString = _
    "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\NWIND.MDB"
    cn.Open()
    MsgBox(cn.ConnectionString)
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    ADODBConnect()
End Sub

Private Sub cn_ConnectComplete( _
    ByVal pError As ADODB.Error, _
    ByRef adStatus As ADODB.EventStatusEnum, _
    ByVal pConnection As ADODB.Connection) _
    Handles cn.ConnectComplete

    '  This is the event handler for the cn_ConnectComplete event raised
    '  by the ADODB.Connection object when a database is opened.
    Dim x As Integer = 6
    Dim y As Integer = 0
    Try
        x = CInt(x / y) ' Attempt to divide by zero.
        ' This procedure would fail silently without exception handling.
    Catch ex As Exception
        MsgBox("There was an error: " & ex.Message)
    End Try
End Sub
```

Cet exemple génère une erreur comme prévu. Cependant, si vous essayez le même exemple sans le bloc **Try...Catch...Finally**, 'erreur est ignorée comme si vous utilisiez l'instruction **OnError Resume Next**. Sans la gestion des erreurs, la division par zéro échoue silencieusement. Étant donné que ces erreurs ne déclenchent jamais d'erreur de type exception non gérée, il est vital que vous utilisiez un certain mode de gestion des exceptions dans les gestionnaires d'événements qui gèrent les événements provenant des objets COM.

Compréhension des erreurs d'interopérabilité COM

Sans la gestion des erreurs, les appels Interop génèrent souvent des erreurs qui fournissent peu d'informations. Autant que possible, utilisez la gestion structurée des erreurs pour fournir de plus amples informations sur les problèmes lorsque ceux-ci se produisent. Cela peut s'avérer particulièrement utile lors du débogage d'applications. Par exemple :

VB

```
Try
    ' Place call to COM object here.
Catch ex As Exception
    ' Display information about the failed call.
End Try
```

Vous pouvez obtenir des informations telles que la description des erreurs, les valeurs HRESULT et la source des erreurs COM par l'examen du contenu de l'objet exception.

Problèmes liés aux contrôles ActiveX

La plupart des contrôles ActiveX qui fonctionnent avec Visual Basic 6.0 acceptent Visual Basic 2005 sans problème. Les principales exceptions sont les contrôles conteneurs, des contrôles qui contiennent visuellement d'autres contrôles. Voici quelques exemples d'anciens contrôles qui ne fonctionnent pas correctement avec Visual Studio :

- le contrôle Frame de Microsoft Forms 2.0 ;
- le contrôle Up-Down, également désigné contrôle Spin ;
- le contrôle Sheridan Tab.

Il n'existe que peu de solutions aux problèmes des contrôles ActiveX non pris en charge. Vous pouvez faire migrer les contrôles existants vers Visual Studio si vous possédez le code source d'origine. Sinon, vous pouvez rechercher auprès des fournisseurs de logiciels les versions de contrôles actualisées compatibles avec .NET pour remplacer les contrôles ActiveX non pris en charge.

Transmission des propriétés ReadOnly de contrôles avec ByRef

Visual Basic 2005 génère parfois des erreurs COM (par exemple, "Error 0x800A017F CTL_E_SETNOTSUPPORTED") lorsque vous tentez de transmettre à d'autres procédures les propriétés **ReadOnly** de certains anciens contrôles ActiveX sous forme de paramètres **ByRef**. Des appels de procédure similaires à partir de Visual Basic 6.0 ne génèrent pas d'erreur et les paramètres sont traités comme si vous les transmettiez par valeur. Le message d'erreur que vous visualisez dans Visual Basic 2005 correspond à l'objet COM qui signale que vous tentez de modifier une propriété sans procédure de propriété **Set**.

Si vous avez accès à la procédure qui est en cours d'appel, vous pouvez éviter cette erreur en employant le mot clé **ByVal** pour déclarer des paramètres qui acceptent les propriétés **ReadOnly**. Par exemple :

VB

```
Sub ProcessParams(ByVal c As Object)
    'Use the arguments here.
End Sub
```

Si vous n'avez pas accès au code source de la procédure qui est en cours d'appel, vous pouvez forcer la transmission de la propriété par valeur. Pour ce faire, encadrez la procédure appelante d'une paire de crochets supplémentaires. Par exemple, dans un projet qui fait référence à l'objet COM de la bibliothèque Microsoft ActiveX Data Objects 2.8, vous pouvez utiliser :

VB

```
Sub PassByVal(ByVal pError As ADODB.Error)
    ' The extra set of parentheses around the arguments
    ' forces them to be passed by value.
    ProcessParams((pError.Description))
End Sub
```

Déploiement d'assemblys exposant l'interopérabilité

Le déploiement d'assemblys exposant des interfaces COM soulève des défis uniques. Ainsi, un problème potentiel se pose lorsque des applications distinctes référencent le même assembly COM. Cette situation est courante lorsqu'une nouvelle

version d'un assembly est installée et qu'une autre application continue d'utiliser l'ancienne version de l'assembly. La désinstallation d'un assembly partageant une DLL peut involontairement rendre cette DLL indisponible aux autres assemblys.

Pour éviter ce problème, vous devez installer les assemblys partagés dans le Global Assembly Cache (GAC) et utiliser un module de fusion pour le composant. Si vous ne pouvez pas installer l'application dans le GAC, vous devez l'installer dans un sous-répertoire spécifique à la version du répertoire CommonFilesFolder.

Les assemblys qui ne sont pas partagés doivent être placés côte à côte dans le répertoire avec l'application appelante.

Voir aussi

Tâches

Procédure pas à pas : implémentation de l'héritage avec les objets COM

Référence

Type Library Importer (Tlbimp.exe)
Type Library Exporter (Tlbexp.exe)
Inherits, instruction
MarshalAsAttribute

Concepts

Modifications des types de données pour les utilisateurs de Visual Basic 6.0
Introduction aux modules de fusion
Projets de modules de fusion
Global Assembly Cache

Autres ressources

COM Interop
Marshaling d'interopérabilité

Interopérabilité COM dans les applications .NET Framework

Avant d'utiliser des objets COM et des objets .NET Framework dans la même application, vous devez résoudre les différences entre ces objets concernant la manière dont ils existent dans la mémoire. Un objet .NET Framework se trouve dans la mémoire managée (la mémoire contrôlée par le Common Language Runtime) et peut être déplacé par le runtime si nécessaire. Un objet COM se trouve dans la mémoire non managée et n'est pas supposé être déplacé à un autre emplacement dans la mémoire. Visual Studio et le .NET Framework fournissent des outils pour contrôler l'interaction de ces composants managé et non managé. Pour plus d'informations sur le code managé, consultez Common Language Runtime.

En plus d'utiliser des objets COM dans les applications .NET, vous pouvez également utiliser Visual Basic pour développer des objets accessibles à partir de code non managé via COM.

Les liens dans cette page fournissent des détails sur les interactions entre les objets COM et les objets .NET Framework.

Rubriques connexes

COM Interop

Propose des liens vers des rubriques traitant de l'interopérabilité COM en Visual Basic, y compris les objets COM, les contrôles ActiveX, les DLL Win32, les objets managés et l'héritage d'objets COM.

Erreur de Wrapper COM Interop

Décrit les conséquences et les options si le système de projet ne peut pas créer un wrapper d'interopérabilité COM pour un composant particulier.

Interopération avec du code non managé

Décrit brièvement certains problèmes d'interaction entre le code managé et le code non managé et propose des liens pour une étude plus approfondie.

Wrappers COM

Présente les wrappers RCW (Runtime Callable Wrapper), qui permettent au code managé d'appeler les méthodes COM, ainsi que les wrappers COM pouvant être appelés, qui permettent aux clients COM d'appeler les méthodes d'objets .NET.

Interopérabilité COM avancée

Propose des liens vers des rubriques traitant de l'interopérabilité COM par rapport aux wrappers, aux exceptions, à l'héritage, au threading, aux événements, aux conversions et au marshaling.

Automatisation d'applications à l'aide du modèle objet Office

Fournit des informations sur l'utilisation de Microsoft Office et de Visual Studio en tant que partie intégrante de votre application d'entreprise.

Type Library Importer (Tlbimp.exe)

Présente l'outil que vous pouvez utiliser pour convertir les définitions de type trouvées dans une bibliothèque de types COM en définitions équivalentes dans un assembly Common Language Runtime.

Multithreading dans Visual Basic

Les applications Visual Basic peuvent exécuter de nombreuses tâches simultanément à l'aide du *multithreading* (ou *traitement dissocié des threads*), processus dans lequel des tâches distinctes s'exécutent sur des threads séparés. Le multithreading améliore les performances et la réactivité de vos applications.

Dans cette section

Applications multithread

Décrit comment créer et utiliser les threads.

États d'un thread

Indique comment identifier et changer l'état d'un thread.

Paramètres et valeurs de retour pour les procédures multithread

Décrit comment transmettre et retourner des paramètres avec des applications multithread.

Multithreading avec des formulaires et contrôles

Aborde des aspects particuliers de l'utilisation du multithreading, des formulaires et des contrôles.

Procédure pas à pas : multithreading

Montre comment créer une application multithread simple.

Rubriques connexes

Multithreading avancé avec Visual Basic

Répertorie les rubriques qui décrivent comment gérer les threads dans les applications multithread

Multithreading dans les composants

Fournit des liens vers des rubriques qui décrivent comment utiliser le multithreading dans la programmation de composant.

Procédure pas à pas : création d'un composant simple multithread avec Visual Basic

Montre comment créer des composants multithread.

Applications multithread

Visual Basic vous permet d'écrire des applications capables d'exécuter simultanément plusieurs tâches. Les tâches ayant la particularité de pouvoir suspendre d'autres tâches peuvent s'exécuter sur des threads séparés ; ce processus est connu sous le nom de *multithreading* ou *traitement dissocié des threads*.

Les applications qui utilisent le multithreading réagissent mieux aux entrées d'utilisateur parce que leur interface utilisateur reste active pendant que les tâches qui sollicitent beaucoup le processeur s'exécutent sur des threads séparés. Le multithreading est également utile lorsqu'il s'agit de créer des applications évolutives, parce qu'il vous est possible d'ajouter des threads en fonction de l'augmentation de la charge de travail.

Utilisation du composant BackgroundWorker

La méthode la plus fiable pour créer une application multithread consiste à utiliser le composant BackgroundWorker. Cette classe gère un thread distinct dédié au traitement de la méthode spécifiée. Pour un exemple, consultez Procédure pas à pas : multithreading.

Pour démarrer une opération en arrière-plan, créez un objet **BackgroundWorker** et écoutez les événements qui indiquent la progression de votre opération et lorsque celle-ci se termine. Vous pouvez créer l'objet **BackgroundWorker** par programme, ou vous pouvez le déplacer dans votre formulaire à partir de l'onglet **Composants** de la **Boîte à outils**. Si vous créez l'objet **BackgroundWorker** dans le **Concepteur Forms**, il apparaît dans la **Barre d'état des composants**, et ses propriétés sont affichées dans la fenêtre **Propriétés**.

Configuration d'une opération d'arrière-plan

Pour configurer une opération d'arrière-plan, ajoutez un gestionnaire d'événements pour l'événement DoWork. Appelez votre opération de longue durée dans ce gestionnaire d'événements.

Pour démarrer l'opération, appelez RunWorkerAsync. Pour recevoir des notifications de mises à jour de progression, gérez l'événement ProgressChanged. Pour recevoir une notification lorsque l'opération est terminée, gérez l'événement RunWorkerCompleted.

Les méthodes qui gèrent les événements **ProgressChanged** et **RunWorkerCompleted** peuvent accéder à l'interface utilisateur de l'application, car ces événements sont déclenchés sur le thread qui a appelé la méthode **RunWorkerAsync**. Toutefois, le gestionnaire d'événements **DoWork** ne peut pas manipuler les objets d'interface utilisateur car il s'exécute sur le thread d'arrière-plan.

Création et utilisation de threads

Si vous avez besoin d'un plus grand contrôle sur le comportement des threads de votre application, vous pouvez gérer vous-même les threads. Toutefois, n'oubliez pas que l'écriture d'applications multithread correctes peut s'avérer difficile : votre application peut cesser de répondre ou rencontrer des erreurs transitoires dues aux conditions de concurrence critique. Pour plus d'informations, consultez Composants thread-safe.

Pour créer un thread en Visual Basic, vous devez déclarer une variable de type Thread et appeler le constructeur avec l'instruction **AddressOf** et le nom de la procédure ou de la méthode que vous voulez exécuter sur le nouveau thread. Le code suivant en est un exemple :

VB

```
Dim TestThread As New System.Threading.Thread(AddressOf TestSub)
```

Démarrage et interruption de threads

Pour démarrer l'exécution d'un nouveau thread, utilisez la méthode Start, comme dans le code suivant :

VB

```
TestThread.Start()
```

Pour arrêter l'exécution d'un thread, utilisez la méthode Abort, comme dans le code suivant :

VB

```
TestThread.Abort()
```

Vous pouvez non seulement démarrer et interrompre des threads, mais aussi les suspendre en appelant la méthode Sleep ou Suspend, faire redémarrer un thread suspendu avec la méthode Resume et détruire un thread à l'aide de la méthode **Abort**, comme dans le code suivant :

VB

```
TestThread.Sleep(1000)
TestThread.Abort()
```

Méthodes de thread

Le tableau suivant présente quelques méthodes qui permettent de contrôler des threads individuels.

Méthode	Action
Start	Déclenche le démarrage d'un thread.
Sleep	Suspend un thread pendant un délai déterminé.
Suspend	Suspend un thread lorsque celui-ci atteint un point sûr.
Abort	Arrête un thread lorsque celui-ci atteint un point sûr.
Resume	Redémarre un thread suspendu.
Join	Force le thread actif à attendre la fin de l'exécution d'un autre thread. Utilisée avec un délai d'attente, cette méthode retourne la valeur **True** si le thread se termine dans les temps impartis.

Points sûrs

La plupart de ces méthodes sont explicatives, mais le concept de *point sûr* peut être nouveau pour vous. Les points sûrs sont des emplacements de code où l'environnement CLR (common language runtime) peut effectuer en toute sécurité une opération automatique de *garbage collection*, qui consiste à libérer des variables inutilisées et à récupérer l'espace mémoire correspondant. Lorsque vous appelez la méthode **Abort** ou **Suspend** d'un thread, l'environnement CLR analyse le code et détermine le point approprié pour l'arrêt du thread.

Propriétés de thread

Les threads contiennent également un certain nombre de propriétés utiles, décrites dans le tableau suivant :

Propriété	Valeur
IsAlive	Contient la valeur **True** si un thread est actif.
IsBackground	Obtient ou définit une valeur Boolean qui indique si un thread est ou doit être un thread d'arrière-plan. Les threads d'arrière-plan ressemblent aux threads de premier plan, mais ils n'empêchent pas l'arrêt d'un processus. Lorsque tous les threads de premier plan appartenant à un processus sont terminés, l'environnement CLR (common language runtime) met fin au processus par l'appel de la méthode **Abort** sur les threads d'arrière-plan restés actifs.
Name	Obtient ou définit le nom d'un thread. Sert le plus souvent à découvrir des threads individuels lors du débogage.
Priority	Obtient ou définit une valeur utilisée par le système d'exploitation pour classer la planification des threads par priorité.

ApartmentState	Obtient ou définit le modèle de thread utilisé pour un thread particulier. Les modèles de thread sont importants lorsqu'un thread appelle du code non managé.
ThreadState	Contient une valeur qui décrit l'état ou les états d'un thread.

Pour plus d'informations sur les états et les méthodes de threads, consultez États d'un thread.

Priorités associées aux threads

Chaque thread a une propriété Priority, qui détermine le temps processeur dont il dispose pour s'exécuter. Plus le degré de priorité est élevé, plus les plages de temps affectées aux threads par le système d'exploitation sont importantes. Les nouveaux threads sont créés avec la valeur Normal, mais vous pouvez affecter à la propriété **Priority** une autre valeur dans l'énumération ThreadPriority.

Consultez **ThreadPriority** pour obtenir une description détaillée des différentes priorités des threads.

Threads de premier plan et d'arrière-plan

Un *thread de premier plan* peut s'exécuter indéfiniment, tandis qu'un *thread d'arrière-plan* prend fin dès l'arrêt du dernier thread de premier plan. Vous pouvez utiliser la propriété **IsBackground** pour déterminer ou changer l'état d'arrière-plan d'un thread.

Voir aussi

Concepts

États d'un thread
Paramètres et valeurs de retour pour les procédures multithread
Synchronisation des threads
Multithreading avec des formulaires et contrôles

Autres ressources

Multithreading dans les composants

États d'un thread

La propriété ThreadState fournit des informations sur l'état d'un thread. Comme les threads peuvent être dans plusieurs états à un même moment, la valeur stockée dans **ThreadState** peut être une combinaison des valeurs de l'énumération ThreadState. Par exemple, si un thread est bloqué sur un appel à Wait et qu'un autre thread appelle la méthode Abort sur ce même thread, le thread sera simultanément à l'état WaitSleepJoin et AbortRequested.

Changement des états d'un thread

Dès le démarrage d'un thread, vous pouvez appeler ses méthodes pour changer son état. Par exemple, vous pouvez provoquer la suspension d'un thread pendant quelques millisecondes en appelant System.Threading.Thread.Sleep(System.Int32). La méthode **Sleep** prend comme paramètre un délai d'expiration qui est le nombre de millisecondes pendant lequel le thread reste bloqué.

L'appel de **Sleep** avec l'argument Infinite entraîne la mise en veille d'un thread jusqu'à son interruption par un autre thread qui appelle Interrupt. La méthode **Interrupt** libère le thread de destination de l'attente dans laquelle il peut se trouver et entraîne la levée d'une exception.

Vous pouvez également suspendre un thread en appelant Suspend. Lorsqu'un thread appelle **Suspend** sur lui-même, l'appel bloque jusqu'à ce qu'un autre thread le reprenne en appelant Resume. Lorsqu'un thread appelle **Suspend** sur un autre thread, l'appel est un appel sans blocage qui entraîne la suspension de l'autre thread. L'appel de **Resume** permet de mettre un terme à la suspension d'un autre thread et lui fait reprendre son exécution. Contrairement à **Sleep**, **Suspend** n'interrompt pas immédiatement un thread ; le thread suspendu ne s'arrête pas tant que le Common Language Runtime n'a pas déterminé qu'il a atteint un point dépourvu de risque.

La méthode **Abort** interrompt un thread en cours d'exécution en déclenchant une exception ThreadAbortException qui entraîne la mort du thread.

Pour plus d'informations sur ces méthodes, consultez Thread.

Voir aussi

Référence

SyncLock, instruction
System.Threading

Concepts

Applications multithread
Synchronisation des threads
Paramètres et valeurs de retour pour les procédures multithread
Multithreading avec des formulaires et contrôles
Délégués et opérateur AddressOf

Autres ressources

Multithreading dans les composants

Paramètres et valeurs de retour pour les procédures multithread

L'apport et le retour des valeurs dans une application multithread se trouvent compliqués par le fait que le constructeur de la classe de thread doit recevoir une référence à une procédure qui n'accepte aucun argument et ne retourne aucune valeur. Les sections ci-dessous indiquent quelques méthodes simples pour fournir des paramètres et retourner des valeurs à partir de procédures sur des threads séparés.

Apport de paramètres pour des procédures multithread

Le meilleur moyen de fournir des paramètres pour un appel de méthode multithread consiste à placer la méthode cible dans une classe et à définir pour cette classe des champs qui vont servir de paramètres pour le nouveau thread. Cette approche présente l'avantage de vous permettre de créer une nouvelle instance de la classe, avec ses propres paramètres, à chaque fois que vous voulez démarrer un nouveau thread. Prenons l'exemple d'une fonction calculant la surface d'un triangle, comme dans le code suivant :

VB

```
Function CalcArea(ByVal Base As Double, ByVal Height As Double) As Double
    CalcArea = 0.5 * Base * Height
End Function
```

Vous pouvez écrire une classe qui contient la fonction `CalcArea` et crée des champs pour stocker les paramètres d'entrée, comme suit :

VB

```
Class AreaClass
    Public Base As Double
    Public Height As Double
    Public Area As Double
    Sub CalcArea()
        Area = 0.5 * Base * Height
        MsgBox("The area is: " & Area)
    End Sub
End Class
```

Pour utiliser `AreaClass`, vous pouvez créer un objet `AreaClass` et définir les propriétés `Base` et `Height` conformément au code suivant :

VB

```
Protected Sub TestArea()
    Dim AreaObject As New AreaClass
    Dim Thread As New System.Threading.Thread _
                      (AddressOf AreaObject.CalcArea)
    AreaObject.Base = 30
    AreaObject.Height = 40
    Thread.Start()
End Sub
```

Vous remarquerez que la procédure `TestArea` ne vérifie pas la valeur du champ `Area` après avoir appelé la méthode `CalcArea`. Comme `CalcArea` s'exécute sur un thread séparé, le paramétrage du champ `Area` n'est pas garanti si vous le vérifiez immédiatement après avoir appelé `Thread.Start`. La section suivante indique une meilleure méthode pour retourner des valeurs à partir de procédures multithread.

Retour de valeurs à partir de procédures multithread

Le retour de valeurs à partir de procédures qui s'exécutent sur des threads séparés est compliqué par le fait que les procédures ne peuvent ni être des fonctions, ni utiliser des arguments **ByRef**. La méthode la plus facile pour retourner des valeurs consiste

à utiliser le composant BackgroundWorker pour gérer vos threads, à déclencher un événement à la fin de la tâche et à traiter les résultats avec un gestionnaire d'événements.

L'exemple ci-dessous retourne une valeur en déclenchant un événement à partir d'une procédure s'exécutant sur un thread séparé :

VB

```
Private Class AreaClass2
    Public Base As Double
    Public Height As Double
    Function CalcArea() As Double
        ' Calculate the area of a triangle.
        Return 0.5 * Base * Height
    End Function
End Class

Private WithEvents BackgroundWorker1 As New System.ComponentModel.BackgroundWorker

Private Sub TestArea2()
    Dim AreaObject2 As New AreaClass2
    AreaObject2.Base = 30
    AreaObject2.Height = 40

    ' Start the asynchronous operation.
    BackgroundWorker1.RunWorkerAsync(AreaObject2)
End Sub

' This method runs on the background thread when it starts.
Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
    ByVal e As System.ComponentModel.DoWorkEventArgs) _
    Handles BackgroundWorker1.DoWork

    Dim AreaObject2 As AreaClass2 = CType(e.Argument, AreaClass2)
    ' Return the value through the Result property.
    e.Result = AreaObject2.CalcArea()
End Sub

' This method runs on the main thread when the background thread finishes.
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
    ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
    Handles BackgroundWorker1.RunWorkerCompleted

    ' Access the result through the Result property.
    Dim Area As Double = CDbl(e.Result)
    MsgBox("The area is: " & Area)
End Sub
```

Vous pouvez fournir des paramètres et des valeurs de retour aux threads ThreadPool à l'aide de la variable objet état **ByVal** (facultative) de la méthode QueueUserWorkItem. Les threads de type composant Timer thread prennent également en charge un objet état à cet effet. Pour plus d'informations sur le regroupement des threads et les composants Timer thread, consultez Regroupement des threads et Composants Timer thread.

Voir aussi

Concepts

États d'un thread
Regroupement des threads
Synchronisation des threads
Événements et gestionnaires d'événements
Applications multithread
Multithreading avec des formulaires et contrôles
Délégués et opérateur AddressOf

Autres ressources

Multithreading dans les composants

Multithreading avec des formulaires et contrôles

Outre le fait que le multithreading soit la solution la mieux adaptée aux procédures en cours d'exécution et aux méthodes de classe, vous pouvez également l'utiliser avec des formulaires et des contrôles. Si vous choisissez de l'utiliser de cette façon, tenez bien compte des points suivants :

- Autant que possible, n'exécutez les méthodes d'un contrôle que sur le thread avec lequel ce contrôle a été créé. Si vous devez appeler une méthode d'un contrôle à partir d'un autre thread, vous devez utiliser Invoke.
- N'utilisez pas l'instruction **SyncLock** pour verrouiller des threads qui manipulent des contrôles ou des formulaires. Comme les méthodes des contrôles et formulaires rappellent parfois une procédure d'appel, vous risquez de créer sans vous en rendre compte un blocage ; dans cette situation, deux threads attendent chacun que l'autre annule le verrou, ce qui entraîne l'arrêt de l'application.

Voir aussi

Référence

Invoke
InvokeRequired

Concepts

États d'un thread
Paramètres et valeurs de retour pour les procédures multithread
Applications multithread

Autres ressources

Multithreading dans les composants

Procédure pas à pas : multithreading

Cette procédure pas à pas montre comment créer une application multithread qui recherche dans un fichier texte les occurrences d'un mot. Elle présente les points suivants :

- Définition d'une classe à l'aide d'une méthode qui peut être appelée par le composant BackgroundWorker.
- Gestion des événements déclenchés par le composant **BackgroundWorker**.
- Démarrage d'un composant **BackgroundWorker** pour l'exécution d'une méthode.
- Implémentation d'un bouton `Cancel` qui arrête le composant **BackgroundWorker**.

Pour créer l'exemple de code de cette rubrique

1. Ouvrez un nouveau projet Application Windows Visual Basic, puis créez un formulaire nommé `Form1`.
2. Ajoutez deux boutons et quatre zones de texte à `Form1`.
3. Nommez les objets de la façon indiquée dans le tableau suivant.

Objet	Propriété	Valeur
Premier bouton	**Name**, **Text**	Start, Start
Deuxième bouton	**Name**, **Text**	Cancel, Cancel
Première zone de texte	**Name**, **Text**	SourceFile, ""
Deuxième zone de texte	**Name**, **Text**	CompareString, ""
Troisième zone de texte	**Name**, **Text**	WordsCounted, "0"
Quatrième zone de texte	**Name**, **Text**	LinesCounted, "0"

4. Ajoutez une étiquette à côté de chaque zone de texte. Définissez la propriété **Text** pour chaque étiquette, comme illustré dans le tableau suivant.

Objet	Propriété	Valeur
Première étiquette	**Text**	Source File
Deuxième étiquette	**Text**	Compare String
Troisième étiquette	**Text**	Matching Words
Quatrième étiquette	**Text**	Lines Counted

5. Ajoutez un composant **BackgroundWorker** contenu dans la section **Composants** de la **Boîte à outils** à votre formulaire. Il apparaît dans la barre d'état des composants du formulaire.
6. Définissez les propriétés suivantes pour l'objet `BackgroundWorker1`.

Propriété	Valeur
WorkerReportsProgress	True
WorkerSupportsCancellation	True

7.

Pour définir la méthode qui s'exécutera sur un thread distinct

1. Dans le menu **Projet**, sélectionnez **Ajouter une classe** pour ajouter une classe au projet. La boîte de dialogue **Ajouter un nouvel élément** s'affiche.
2. Sélectionnez **Classe** dans la fenêtre des modèles et tapez `Words.vb` dans le champ du nom.
3. Cliquez sur **Ajouter**. La classe `Words` s'affiche.
4. Ajoutez une instruction **Option Compare** en haut de la classe `Words`, au-dessus de l'instruction **Class** :

   VB

   ```
   Option Compare Text ' Case insensitive search.
   ' Use Option Compare Binary for case sensitive search.
   ```

5. Ajoutez le code suivant à la classe `Words` :

   VB

   ```
   Public Class Words
       ' Object to store the current state, for passing to the caller.
       Public Class CurrentState
           Public LinesCounted As Integer
           Public WordsMatched As Integer
       End Class

       Public SourceFile As String
       Public CompareString As String
       Private WordCount As Integer = 0
       Private LinesCounted As Integer = 0

       Public Sub CountWords( _
           ByVal worker As System.ComponentModel.BackgroundWorker, _
           ByVal e As System.ComponentModel.DoWorkEventArgs _
       )
           ' Initialize the variables.
           Dim mystream As System.IO.StreamReader = Nothing
           Dim line As String = ""

           If CompareString Is Nothing Or _
                   CompareString = System.String.Empty Then
               Throw New Exception("CompareString not specified.")
           End If

           Try
               ' Open a new stream.
               mystream = My.Computer.FileSystem.OpenTextFileReader(SourceFile)
               ' Do until the stream returns Nothing at end of file.
               Do Until line Is Nothing
                   If worker.CancellationPending Then
                       e.Cancel = True
                       Exit Do
                   Else
                       line = mystream.ReadLine
                       WordCount += CountInString(line, CompareString)
                       LinesCounted += 1 ' Increment line count.

                       ' Raise an event so the form can monitor progress.
                       Dim state As New CurrentState
   ```

```
                    state.LinesCounted = LinesCounted
                    state.WordsMatched = WordCount
                    worker.ReportProgress(0, state)
                End If
            Loop
        Finally
            If mystream IsNot Nothing Then
                ' Close the file.
                mystream.Close()
            End If
        End Try
    End Sub

    Private Function CountInString( _
        ByVal SourceString As String, _
        ByVal CompareString As String _
    ) As Integer
        ' This function counts the number of times
        ' a word is found in a line.
        If SourceString Is Nothing Then
            Return 0
        End If

        Dim regex As New System.Text.RegularExpressions.Regex( _
            System.Text.RegularExpressions.Regex.Escape(CompareString))
        Dim matches As System.Text.RegularExpressions.MatchCollection
        matches = regex.Matches(SourceString)
        Return matches.Count
    End Function
End Class
```

Pour gérer les événements à partir du thread

- Ajoutez les gestionnaires d'événements suivants au formulaire principal :

  VB

```
Private Sub BackgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
    Handles BackgroundWorker1.RunWorkerCompleted

    ' This event handler is called when the background thread finishes.
    ' This method runs on the main thread.

    If e.Error IsNot Nothing Then
        MsgBox("Error: " & e.Error.Message)
    ElseIf e.Cancelled Then
        MsgBox("Word counting canceled.")
    Else
        MsgBox("Finished counting words.")
    End If
End Sub

Private Sub BackgroundWorker1_ProgressChanged( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
    Handles BackgroundWorker1.ProgressChanged
```

```
    ' This event handler is called after the background thread
    ' reads a line from the source file.
    ' This method runs on the main thread.

    Dim state As Words.CurrentState = _
        CType(e.UserState, Words.CurrentState)
    Me.LinesCounted.Text = state.LinesCounted.ToString
    Me.WordsCounted.Text = state.WordsMatched.ToString
End Sub
```

Pour démarrer et appeler un nouveau thread qui exécute la méthode WordCount

1. Ajoutez les procédures suivantes à votre programme :

   VB

```
Private Sub BackgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.DoWorkEventArgs) _
    Handles BackgroundWorker1.DoWork

    ' This event handler is where the actual work is done.
    ' This method runs on the background thread.

    ' Get the BackgroundWorker object that raised this event.
    Dim worker As System.ComponentModel.BackgroundWorker
    worker = CType(sender, System.ComponentModel.BackgroundWorker)

    ' Get the Works object and call the main method.
    Dim WC As Words = CType(e.Argument, Words)
    WC.CountWords(worker, e)
End Sub

Sub StartThread()
    ' This method runs on the main thread.

    Me.WordsCounted.Text = "0"

    ' Initialize the object that the background worker calls.
    Dim WC As New Words
    WC.CompareString = Me.CompareString.Text
    WC.SourceFile = Me.SourceFile.Text

    ' Start the asynchronous operation.
    BackgroundWorker1.RunWorkerAsync(WC)
End Sub
```

2. Appelez la méthode `StartThread` à partir du bouton `Start` sur le formulaire :

   VB

```
Private Sub Start_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Start.Click

    StartThread()
```

```
End Sub
```

Pour implémenter un bouton Cancel qui arrête le thread

- Appelez la procédure `StopThread` à partir du gestionnaire d'événements `Click` pour le bouton `Cancel`.

  VB

```
Private Sub Cancel_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Cancel.Click

    ' Cancel the asynchronous operation.
    Me.BackgroundWorker1.CancelAsync()
End Sub
```

Test

Vous pouvez à présent tester l'application afin de vous assurer qu'elle fonctionne correctement.

Pour tester l'application

1. Appuyez sur F5 pour exécuter l'application.
2. Quand le formulaire s'affiche, entrez le chemin d'accès à un fichier de votre choix pour réaliser le test dans la zone `sourceFile`. Par exemple, supposons que le fichier de test se nomme Test.txt, entrez C:\Test.txt.
3. Dans la deuxième zone de texte, entrez un mot ou une expression que l'application va devoir rechercher dans le fichier texte.
4. Cliquez sur le bouton `Start`. Sur le bouton `LinesCounted`, la valeur affichée devrait commencer immédiatement à être incrémentée. L'application affiche le message « Finished Counting » une fois terminé.

Pour tester le bouton Cancel

1. Appuyez sur F5 pour démarrer l'application, puis entrez le nom de fichier et le mot à rechercher comme le décrit la procédure précédente. Assurez-vous que le fichier choisi est suffisamment volumineux pour que vous ayez le temps d'annuler la procédure avant qu'elle ne soit terminée.
2. Cliquez sur le bouton `Start` pour lancer l'application.
3. Cliquez sur le bouton `Cancel`. L'application devrait cesser le décompte immédiatement.

Étapes suivantes

Cette application contient une fonctionnalité de base de gestion des erreurs. Elle détecte les chaînes de recherche vides. Vous pouvez ajouter robustesse à ce programme en lui permettant de gérer d'autres erreurs, telles que le dépassement du nombre maximal de mots ou de lignes pouvant être dénombrés.

Voir aussi

Tâches

Procédure pas à pas : création d'un composant simple multithread avec Visual Basic

Autres ressources

Multithreading dans Visual Basic

Référence (Visual Basic)

Cette section vous renvoie à des informations de référence concernant divers aspects de la programmation Visual Basic.

Dans cette section

Référence Visual Basic

Fournit des informations de référence sur différents aspects du langage Visual Basic.

Outils de ligne de commande de Visual Basic

Décrit le compilateur de ligne de commande Visual Basic, les options du compilateur et l'outil de mise à niveau de mot clé.

Mise à niveau de la référence pour Visual Basic

Fournit des liens vers des informations sur la mise à niveau de vos applications à partir de versions antérieures de Visual Basic vers la version actuelle.

Informations de référence du .NET Framework

Fournit des liens vers des informations sur l'utilisation de la bibliothèque de classes .NET Framework.

Spécification du langage Visual Basic

Fournit des liens vers la spécification du langage Visual Basic complète qui contient des informations détaillées sur tous les aspects du langage.

Rubriques connexes

Éléments généraux de l'interface utilisateur (Visual Studio)

Contient des rubriques relatives aux boîtes de dialogue et fenêtres utilisées dans Visual Studio.

Références de schémas XML

Fournit des liens vers des rubriques relatives aux différents schémas XML disponibles dans Visual Studio.

Équivalents de langage

Compare les mots clés, les concepts de programmation, les types de données, les opérateurs, les contrôles et les objets programmables dans plusieurs langages de programmation.

Guide de référence de l'extensibilité et de l'automation

Fournit des liens vers des rubriques qui traitent de l'automation et de l'extensibilité dans Visual Studio, à la fois pour les composants partagés et spécifiques à la langue.

Référence Visual Basic

Cette section fournit des informations de référence sur différents aspects du langage Visual Basic.

Dans cette section

Conventions typographiques

Récapitule la façon dont les mots clés, les espaces réservés et d'autres éléments du langage sont mis en forme dans la documentation Visual Basic.

Mots clés du langage Visual Basic

Documente les mots clés réservés et non réservés du langage Visual Basic.

Membres de la bibliothèque runtime Visual Basic

Répertorie les classes et modules de l'espace de noms Microsoft.VisualBasic, avec des liens vers leurs fonctions, méthodes, propriétés, constantes et énumérations membres.

Mots clés et membres par tâche

Récapitule les mots clés du langage Visual Basic et les membres de la bibliothèque Runtime organisés selon leur finalité et leur utilisation.

Attributs Visual Basic

Documente les attributs disponibles dans Visual Basic.

Constantes et énumérations (Visual Basic)

Documente les constantes et énumérations disponibles dans Visual Basic.

Liste des types de données (Visual Basic)

Documente les types de données disponibles dans Visual Basic.

Directives (Visual Basic)

Documente les directives du compilateur disponibles dans Visual Basic.

Fonctions (Visual Basic)

Documente les fonctions runtime disponibles dans Visual Basic.

Mots clés (Visual Basic)

Documente les mots clés du langage disponibles dans Visual Basic.

Modules (Visual Basic)

Documente les modules disponibles dans Visual Basic et leurs membres.

Objets (Visual Basic)

Documente les objets disponibles dans Visual Basic et leurs membres.

Opérateurs (Visual Basic)

Documente les opérateurs disponibles dans Visual Basic.

Propriétés (Visual Basic)

Documente les propriétés disponibles dans Visual Basic.

Instructions (Visual Basic)

Documente la déclaration et les instructions d'exécutable disponibles dans Visual Basic.

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Décrit les commentaires de documentation pour lesquels IntelliSenseTM est fourni dans l'éditeur de code Visual Basic.

Messages d'erreur (Visual Basic)

Fournit la liste des messages d'erreur du compilateur et d'exécution Visual Basic et vous aide à les gérer.

Rubriques connexes

Visual Basic

Offre une aide complète de tous les domaines du langage Visual Basic.

Compilateur Visual Basic

Décrit comment utiliser le compilateur de ligne de commande comme alternative aux programmes de compilation dans l'environnement de développement intégré (IDE) Visual Studio.

Conventions typographiques

La documentation Visual Basic utilise les conventions typographiques et de code suivantes.

Conventions typographiques

Convention	Description
Sub, **If**, **ChDir**, **Print**, **True**, **Debug**	Les mots en gras dont la première lettre est une majuscule indiquent des mots clés spécifiques au langage ou des membres du runtime.
Module, instruction	Les mots et phrases soulignés et affichés dans une police de couleur sont des liens hypertexte sur lesquels vous pouvez cliquer pour accéder immédiatement à une page d'aide correspondante.
object, *variablename*, *argumentlist*	Les mots en italique indiquent des espaces réservés pour fournir des informations.
[Shadows], [*expressionlist*]	Dans la syntaxe, les éléments entre crochets sont facultatifs.
{ **Public** | **Friend** | **Private** }	Dans la syntaxe, les éléments entre accolades et séparés par une barre verticale indiquent un choix obligatoire entre plusieurs éléments. Vous devez choisir un seul de ces éléments.
[**Protected** | **Friend**]	Dans la syntaxe, les crochets et les barres verticales indiquent plusieurs options indépendantes. Vous pouvez choisir une combinaison de ces éléments, ou aucun de ceux-ci.
[{ **ByVal** | **ByRef** }]	Dans la syntaxe, des crochets entourant des accolades et des barres verticales indiquent que vous ne pouvez choisir qu'un seul des éléments, mais que vous pouvez également les ignorer complètement.
membername $_1$, *membername*$_2$, *membername*3	Les indices différencient plusieurs instances d'un même espace réservé.
memerbname1 ... *membernamen*	Dans la syntaxe, les points de suspension indiquent un nombre indéfini d'éléments du type précédant immédiatement ces points de suspension. Dans le code, les points de suspension indiquent que du code a été omis par souci de clarté.
ESC, ENTER	Les mots en majuscules indiquent des noms de touches et des séquences de touches sur votre clavier.
ALT+F1, CTRL+R	Un signe plus (+) placé entre des noms de touches indique une combinaison de touches. Par exemple, ALT+F1 signifie maintenir la touche ALT enfoncée, tout en appuyant sur la touche F1.

Conventions de code

Exemple de code	Description
`sampleString = "Hello, world!"` L'instruction précédente affecte à `sampleString` la valeur « Hello, world! ».	Une police non proportionnelle est utilisée pour les exemples de code, ainsi que pour les éléments de code contenus dans un texte explicatif.
`' This is a comment.` `REM This is also a comment.`	Une apostrophe (') ou le mot clé REM introduisent des commentaires de code.

`sampleVar = "This is an " _`
`& "example" _`
`& " of how to continue code."`
	Un espace suivi d'un trait de soulignement (_) à la fin d'une ligne continue l'instruction de code source.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Mots clés du langage Visual Basic

Membres de la bibliothèque runtime Visual Basic

Concepts

Conventions d'affectation de noms Visual Basic

Commentaires dans le code

Autres ressources

Référence Visual Basic

Mots clés du langage Visual Basic

Les tableaux suivants répertorient tous les mots clés du langage Visual Basic.

Mots clés réservés

Les mots clés suivants sont *réservés*, ce qui signifie que vous ne pouvez pas les utiliser en tant que noms pour vos éléments de programmation, notamment pour vos variables ou vos procédures. Vous pouvez contourner cette restriction en plaçant le nom entre parenthèses (**[ ]**). Pour plus d'informations, consultez la section « Noms d'échappement » de la rubrique Noms d'éléments déclarés.

Remarque
L'utilisation de noms échappés n'est pas recommandée, parce qu'elle peut rendre votre code difficile à lire, et conduire à des erreurs subtiles, difficiles à trouver.

AddHandler	AddressOf	Alias	And
AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar
CDate	CDec	CDbl	Char
CInt	Class	CLng	CObj
Const	Continue	CSByte	CShort
CSng	CStr	CType	CUInt
CULng	CUShort	Date	Decimal
Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each
Else	ElseIf	End	**EndIf**
Enum	Erase	Error	Event
Exit	False	Finally	For
Friend	Function	Get	GetType
Global	**GoSub**	GoTo	Handles
If	Implements	Imports	In
Inherits	Integer	Interface	Is
IsNot	**Let**	Lib	Like

Long	Loop	Me	Mod
Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Narrowing	New
Next	Not	Nothing	NotInheritable
NotOverridable	Object	Of	On
Operator	Option	Optional	Or
OrElse	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Propriété
Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume
Return	SByte	Select	Set
Shadows	Shared	Short	Single
Static	Step	Stop	String
Structure	Sub	SyncLock	Then
Throw	To	True	Try
TryCast	TypeOf	**Variant**	**Wend**
UInteger	ULong	UShort	Using
When	While	Widening	With
WithEvents	WriteOnly	Xor	#Const
#Else	#ElseIf	#End	#If
-	&	&=	*
*=	/	/=	\
\=	^	^=	+
+=	=	-=	

Remarque

EndIf, **GoSub**, **Let**, **Variant** et **Wend** sont conservés en tant que mots clés réservés, bien qu'ils ne soient plus utilisés dans Visual Basic.

Mots clés non réservés

Les mots clés suivants ne sont pas réservés, ce qui signifie que vous pouvez les utiliser en tant que noms pour vos éléments de

programmation. Toutefois, cette tactique n'est pas recommandée, parce qu'elle peut rendre votre code difficile à lire, et conduire à des erreurs subtiles, difficiles à trouver.

Ansi	Assembly	Auto	Binary
Compare	Custom	Explicit	IsFalse
IsTrue	Mid	Off	Preserve
Strict	Text	Unicode	Until
#ExternalSource	#Region		

Voir aussi

Référence

Membres de la bibliothèque runtime Visual Basic

Mots clés et membres par tâche

Comparaison des mots clés dans différents langages

Membres de la bibliothèque runtime Visual Basic

L'espace de noms **Microsoft.VisualBasic** contient les classes, les modules, les constantes et les énumérations qui constituent la bibliothèque runtime Visual Basic. Ces membres de bibliothèque fournissent les procédures, les propriétés et les valeurs constantes que vous pouvez utiliser dans votre code. Chaque module et chaque classe représentent une catégorie particulière de fonctionnalité.

Classe Microsoft.VisualBasic.Collection

Add	Clear	Contains	Count
GetEnumerator	Item	Remove	

Classe Microsoft.VisualBasic.ComClassAttribute

ClassID	EventID	InterfaceID	InterfaceShadows

Classe Microsoft.VisualBasic.ControlChars

Back	Cr	CrLf	FormFeed
Lf	NewLine	NullChar	Quote
Tab	VerticalTab		

Module Microsoft.VisualBasic.Conversion

ErrorToString	Fix	Hex	Int
Oct	Str	Val	

Module Microsoft.VisualBasic.DateAndTime

DateAdd	DateDiff	DatePart	DateSerial
DateString	DateValue	Day	Hour
Minute	Month	MonthName	Now
Second	TimeOfDay	Timer	TimeSerial
TimeString	TimeValue	Today	Weekday
WeekDayName	Year		

Classe Microsoft.VisualBasic.ErrObject

Clear	Description	Erl	GetException
HelpContext	HelpFile	LastDLLError	Number
Raise	Source		

Module Microsoft.VisualBasic.FileSystem

ChDir	ChDrive	CurDir	Dir
EOF	FileAttr	FileClose	FileCopy
FileDateTime	FileGet	FileGetObject	FileLen

FileOpen	FilePut	FilePutObject	FileWidth
FreeFile	GetAttr	Input	InputString
Kill	LineInput	Loc	Lock
LOF	MkDir	Print	PrintLine
Rename	Reset	RmDir	Seek
SetAttr	SPC	TAB	Unlock
Write	WriteLine		

Module Microsoft.VisualBasic.Financial

DDB	FV	IPmt	IRR
MIRR	NPer	NPV	Pmt
PPmt	PV	Rate	SLN
SYD			

Module Microsoft.VisualBasic.Globals

ScriptEngine	ScriptEngineBuildVersion	ScriptEngineMajorVersion	ScriptEngineMinorVersion

Module Microsoft.VisualBasic.Information

Err	IsArray	IsDate	IsDBNull
IsError	IsNothing	IsNumeric	IsReference
LBound	QBColor	RGB	SystemTypeName
TypeName	UBound	VarType	VbTypeName

Module Microsoft.VisualBasic.Interaction

AppActivate	Beep	CallByName	Choose
Command	CreateObject	DeleteSetting	Environ
GetAllSettings	GetObject	GetSetting	IIf
InputBox	MsgBox	Partition	SaveSetting
Shell	Switch		

Module Microsoft.VisualBasic.Strings

Asc	AscW	Chr	ChrW
Filter	Format	FormatCurrency	FormatDateTime
FormatNumber	FormatPercent	GetChar	InStr

InStrRev	Join	LCase	Left
Len	LSet	LTrim	Mid
Replace	Right	RSet	RTrim
Space	Split	StrComp	StrConv
StrDup	StrReverse	Trim	UCase

Classe Microsoft.VisualBasic.VBFixedArrayAttribute

Bounds	Length		

Classe Microsoft.VisualBasic.VBFixedStringAttribute

Length			

Module Microsoft.VisualBasic.VbMath

Randomize	Rnd		

Constantes et énumérations Microsoft.VisualBasic

L'espace de noms **Microsoft.VisualBasic** fournit des constantes et des énumérations en tant qu'éléments de la bibliothèque Runtime Visual Basic. Vous pouvez utiliser ces valeurs constantes dans votre code. Chaque énumération représente une catégorie particulière de fonctionnalité. Pour plus d'informations, consultez Constantes et énumérations (Visual Basic).

Voir aussi

Référence

Mots clés du langage Visual Basic

Mots clés et membres par tâche

Autres ressources

Espace de noms de runtime Visual Basic dans Visual Studio

Constantes et énumérations (Visual Basic)

Mots clés et membres par tâche

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Category	Description
Liste des tableaux	Création, définition et utilisation de tableaux.
Liste des objets Collection	Utilisation de collections.
Liste des directives du compilateur	Contrôle du comportement du compilateur.
Liste des flux de contrôle	Bouclage et contrôle du flux de procédure.
Liste des conversions	Conversion de nombres et de types de données.
Liste des types de données	Types de données et sous-types d'objets.
Liste des dates et heures	Conversion et utilisation d'expressions de date et d'heure.
Liste des déclarations et des constantes	Déclaration d'éléments de programmation et définition de constantes.
Liste des répertoires et des fichiers	Contrôle du système de fichiers et traitement de ceux-ci.
Liste des erreurs	Interception et retour de valeurs d'erreur.
Liste des mots clés financiers	Exécution de calculs financiers.
Liste des entrées et sorties	Réception d'entrées et affichage ou impression de sorties.
Liste des informations et des interactions	Démarrage d'autres applications et traitement d'événements.
Liste des mots clés mathématiques	Exécution de calculs trigonométriques et autres calculs mathématiques.
Référence My	Documente **My**, une fonctionnalité qui permet un développement facile et rapide de s applications en fournissant un accès intuitif aux méthodes couramment utilisées, a ux propriétés et aux événements utilisés pour la programmation de l'ordinateur sur l equel l'application s'exécute, à l'application actuelle, aux ressources de l'application, a ux paramètres de l'application, etc.
Liste des opérateurs	Comparaison d'expressions et autres opérations.
Liste des registres	Suppression, lecture et enregistrement de paramètres de programme.
Liste des manipulations de chaînes	Manipulation de chaînes et de données de type chaîne.

Voir aussi

Référence

Mots clés du langage Visual Basic

Membres de la bibliothèque runtime Visual Basic

Comparaison des mots clés dans différents langages

Liste des tableaux

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Vérifier un tableau.	IsArray
Déclarer et initialiser un tableau.	Dim, Private, Public, ReDim
Rechercher les limites d'un tableau.	LBound, UBound
Réinitialiser un tableau.	Erase, ReDim

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des objets Collection

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Créez un objet **Collection**.	Collection
Ajouter un élément à une collection.	Add
Enlever un objet d'une collection.	Remove
Référencer un élément d'une collection.	Item
Retourner une référence à une interface IEnumerator.	GetEnumerator

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des directives du compilateur

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Action	Élément du langage
Définir une constante du compilateur.	#Const, directive
Compiler des blocs de code sélectionnés.	#If...Then...#Else, directives
Réduire et masquer des sections de code.	#Region, directive
Indiquer un mappage entre des lignes sources et du texte extérieur à la source.	#ExternalSource, directive

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des flux de contrôle

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Effectuer un branchement.	GoTo, On Error
Quitter ou suspendre le programme.	End, Exit, Stop
Boucler.	Do...Loop, For...Next, For Each...Next, While...End While, With
Prendre des décisions.	Choose, If...Then...Else, Select...Case, Switch
Utiliser des procédures.	Call, Function, Property, Sub

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des conversions

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Convertir une valeur ANSI en chaîne.	Chr, ChrW
Convertir une chaîne en minuscules ou en majuscules.	Format, LCase, UCase
Convertir la date en numéro de série.	DateSerial, DateValue
Convertir un nombre décimal en d'autres bases.	Hex, Oct
Convertir un nombre en chaîne.	Format, Str
Convertir un type de données en un autre.	CBool, CByte, CDate, CDbl, CDec, CInt, CLng, CSng, CShort, CStr, CType, Fix, Int
Convertir la date en jour, mois, jour de la semaine ou année.	Day, Month, Weekday, Year
Convertir l'heure en heure, minute ou seconde.	Hour, Minute, Second
Convertir une chaîne en valeur ASCII.	Asc, AscW
Convertir une chaîne en nombre.	Val
Convertir l'heure en numéro de série.	TimeSerial, TimeValue

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des types de données

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Procéder à une conversion entre des types de données différents	CBool, CByte, CChar, CDate, CDbl, CDec, CInt, CLng, CObj, CShort, CSng, CStr, Fix, Int
Définir des types de données intrinsèques	Boolean, Byte, Char, Date, Decimal, Double, Integer, Long, Object, Short, Single, String
Vérifier des types de données	IsArray, IsDate, IsDbNull, IsError, IsNothing, IsNumeric, IsReference

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des dates et heures

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Obtenir la date ou l'heure actuelle.	Now, Today, TimeOfDay
Effectuer des calculs de date.	DateAdd, DateDiff, DatePart
Retourner une date.	DateSerial, DateValue, MonthName, WeekDayName
Retourner une heure.	TimeSerial, TimeValue
Définir la date ou l'heure.	DateString, TimeOfDay, TimeString, Today
Mesurer la durée d'un processus.	Timer

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des déclarations et des constantes

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Assigner une valeur.	Get, Property
Déclarer des variables ou des constantes.	Const, Dim, Private, Protected, Public, Shadows, Shared, Static
Déclarer une classe, un délégué, une énumération, un module, un espace de noms ou une structure.	Class, Delegate, Enum, Module, Namespace, Structure
Créer des objets.	CreateObject, GetObject, New
Obtenir des informations sur un objet.	GetType, IsArray, IsDate, IsDbNull, IsError, IsNothing, IsNumeric, IsReference, SystemTypeName, TypeName, VarType, VbTypeName
Faire référence à l'objet en cours.	Me
Exiger des déclarations de variables explicites.	Option Explicit, Option Strict
Gérer des événements.	AddHandler, Event, RaiseEvent, RemoveHandler
Implémenter l'héritage.	Inherits, MustInherit, MustOverride, MyBase, MyClass, New, NotInheritable, NotOverridable, Overloads, Overridable, Overrides

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des répertoires et des fichiers

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

L'utilisation de la fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que l'utilisation de ces membres. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

Action	Élément du langage
Changer de répertoire ou de dossier.	ChDir
Changer de lecteur.	ChDrive
Copier un fichier.	FileCopy
Créer un répertoire ou un dossier.	MkDir
Supprimer un répertoire ou un dossier.	RmDir
Renommer un fichier, un répertoire ou un dossier.	Rename
Retourner le chemin d'accès en cours.	CurDir
Retourner les informations de date et d'heure d'un fichier.	FileDateTime
Retourner les attributs de fichier, répertoire ou étiquette.	GetAttr
Retourner la longueur d'un fichier.	FileLen
Retourner le nom d'un fichier ou le nom d'un volume.	Dir
Définir les informations relatives aux attributs d'un fichier.	SetAttr

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic
Création, suppression et déplacement de fichiers et de répertoires en Visual Basic
Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Liste des erreurs

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Générer des erreurs d'exécution.	Clear, Error, Raise
Obtenir des exceptions.	GetException
Fournir des informations relatives à l'erreur.	Err
Intercepter les erreurs au cours de l'exécution.	On Error, Resume, Try...Catch...Finally
Fournir le numéro de ligne de l'erreur.	Erl
Fournir le code de l'erreur système.	LastDLLError

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des mots clés financiers

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Calculer l'amortissement.	DDB, SLN, SYD
Calculer la valeur future.	FV
Calculer le taux d'intérêt.	Rate
Calculer le taux de rendement interne.	IRR, MIRR
Calculer le nombre de périodes.	NPer
Calculer les paiements.	IPmt, Pmt, PPmt
Calculer la valeur actuelle.	NPV, PV

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des informations et des interactions

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Exécuter d'autres programmes.	AppActivate, Shell
Appeler une méthode ou une propriété.	CallByName
Émettre un signal sonore à partir de l'ordinateur.	Beep
Fournir une chaîne de ligne de commande.	Command
Manipuler des objets COM.	CreateObject, GetObject
Récupérer des informations sur la couleur.	QBColor, RGB
Contrôler des boîtes de dialogue.	InputBox, MsgBox

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des entrées et sorties

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Accéder à un fichier ou en créer un.	FileOpen
Fermer des fichiers.	FileClose, Reset
Contrôler l'apparence de la sortie.	Format, Print, SPC, TAB, FileWidth
Copier un fichier.	FileCopy
Obtenir des informations sur un fichier.	EOF, FileAttr, FileDateTime, FileLen, FreeFile, GetAttr, Loc, LOF, Seek
Obtient ou fournit des informations à l'utilisateur grâce à une boîte de dialogue de contrôle.	InputBox, MsgBox
Gérer des fichiers.	Dir, Kill, Lock, Unlock
Lire à partir d'un fichier.	FileGet, FileGetObject, Input, InputString, LineInput
Retourner la longueur d'un fichier.	FileLen
Définir ou obtenir des attributs de fichier.	FileAttr, GetAttr, SetAttr
Définir une position de lecture/écriture dans un fichier.	Seek
Écrire dans un fichier.	FilePut, FilePutObject, Print, Fonctions Write, WriteLine, Fonctions Write, WriteLine

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des mots clés mathématiques

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Action	Élément du langage
Dériver des fonctions trigonométriques.	Atn, Cos, Sin, Tan
Calculs généraux.	Exp, Log, Sqr
Générer des nombres aléatoires.	Randomize, Rnd
Obtenir une valeur absolue.	Abs
Obtenir le signe d'une expression.	Sgn
Effectuer des conversions numériques.	Fix, Int

Voir aussi

Référence

Fonctions mathématiques dérivées (Visual Basic)
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Fonctions mathématiques dérivées (Visual Basic)

Voici une liste de fonctions mathématiques non intrinsèques pouvant être dérivées des fonctions mathématiques intrinsèques :

Fonction	**Équivalents**
Sécante (Sec(x))	*1 / Cos(x)*
Cosécante (Csc(x))	*1 / Sin(x)*
Cotangente (Ctan(x))	*1 / Tan(x)*
Arcsinus (Asin(x))	*Atan(x / Sqrt(-x * x + 1))*
Arccosinus (Acos(x))	*Atan(-x / Sqrt(-x * x + 1)) + 2 * Atan(1)*
Arcsécante (Asec(x))	*2 * Atan(1) – Atan(Sign(x) / Sqrt(x * x – 1))*
Arccosécante (Acsc(x))	*Atan(Sign(x) / Sqrt(x * x – 1))*
Arccotangente (Acot(x))	*2 * Atan(1) - Atan(x)*
Arcsinus hyperbolique (Sinh(x))	*(Exp(x) – Exp(-x)) / 2*
Cosinus hyperbolique (Cosh(x))	*(Exp(x) + Exp(-x)) / 2*
Tangente hyperbolique (Tanh(x))	*(Exp(x) – Exp(-x)) / (Exp(x) + Exp(-x))*
Sécante hyperbolique (Sech(x))	*2 / (Exp(x) + Exp(-x))*
Cosécante hyperbolique (Csch(x))	*2 / (Exp(x) – Exp(-x))*
Cotangente hyperbolique (Coth(x))	*(Exp(x) + Exp(-x)) / (Exp(x) – Exp(-x))*
Arcsinus hyperbolique (Asinh(x))	*Log(x + Sqrt(x * x + 1))*
Arccosinus hyperbolique (Acosh(x))	*Log(x + Sqrt(x * x – 1))*
Arctangente hyperbolique (Atanh(x))	*Log((1 + x) / (1 – x)) / 2*
Arcsécante hyperbolique (AsecH(x))	*Log((Sqrt(-x * x + 1) + 1) / x)*
Arccosécante hyperbolique (Acsch(x))	*Log((Sign(x) * Sqrt(x * x + 1) + 1) / x)*
Arccotangente hyperbolique (Acoth(x))	*Log((x + 1) / (x – 1)) / 2*

Configuration requise

Espace de noms : Math

Assembly : mscorlib (dans mscorlib.dll)

Voir aussi

Référence

Fonctions mathématiques (Visual Basic)

Référence My

La fonctionnalité **My** permet une programmation plus rapide et plus facile grâce à un accès intuitif aux méthodes, aux propriétés et aux événements couramment utilisés. Ce tableau répertorie les objets contenus dans **My** ainsi que les actions qui peuvent être exécutées avec chacun de ces objets.

Action	**Objet**
Accès aux informations d'application et aux services.	My.Application, objet
Accès à l'ordinateur hôte et à ses ressources, services et données.	My.Computer, objet
Accès aux formulaires dans le projet actuel.	My.Forms, objet
Accès au journal d'applications.	My.Log, objet
Accès à la demande Web actuelle.	My.Request, objet
Accédé aux éléments de ressource.	My.Resources, objet
Accès à la réponse Web actuelle.	My.Response, objet
Accès aux paramètres utilisateur et aux paramètres de niveau de l'application.	My.Settings, objet
Accès au contexte de sécurité de l'utilisateur actuel.	My.User, objet
Accès aux services Web XML référencés par le projet actuel.	My.WebServices, objet

Voir aussi

Concepts

Nouveautés de Visual Basic

Vue d'ensemble du modèle d'application Visual Basic

Développement avec My

Liste des opérateurs

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Tâche	Élément du langage
Arithmétique	^, –, *, /, \, Mod, +, =
Assignation	=, ^=, *=, /=, \=, +=, -=, &=
Comparaison	=, <>, <, >, <=, >=, Like, Is
Concaténation	&, +
Opérations logiques/de bits	Not, And, Or, Xor, AndAlso, OrElse
Opérations diverses	AddressOf, GetType

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Liste des registres

Les mots clés du langage Visual Studio et les membres de la bibliothèque runtime sont organisés selon leur objectif et leur utilisation.

L'utilisation de la fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations de Registre que ces éléments. Pour plus d'informations, consultez My.Computer.Registry, objet.

Action	Élément du langage
Supprimer des paramètres du programme.	DeleteSetting
Lire les paramètres du programme.	GetSetting, GetAllSettings
Enregistrer les paramètres du programme.	SaveSetting

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Concepts

Tâches courantes relatives au Registre

Liste des manipulations de chaînes

Les mots clés du langage Visual Basic et les membres de la bibliothèque runtime sont organisés selon leur finalité et leur utilisation.

Action	Élément du langage
Comparer deux chaînes.	StrComp
Convertir des chaînes.	StrConv
Inverser une chaîne.	InStrRev, StrReverse
Convertir en minuscules ou en majuscules.	Format, LCase, UCase
Créer une chaîne de caractères répétés.	Space, StrDup
Trouver la longueur d'une chaîne.	Len
Mettre en forme une chaîne.	Format, FormatCurrency, FormatDateTime, FormatNumber, FormatPercent
Manipuler des chaînes.	InStr, Left, LTrim, Mid, Right, RTrim, Trim
Définir des règles de comparaison de chaînes.	Option Compare
Travailler avec des valeurs ASCII et ANSI.	Asc, AscW, Chr, ChrW
Remplacer une sous-chaîne spécifiée.	Replace
Retourner un tableau de chaînes basé sur un filtre.	Filter
Retourner un nombre spécifié de sous-chaînes.	Split, Join

Voir aussi

Référence

Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Attributs Visual Basic

Visual Basic fournit plusieurs attributs qui permettent aux objets d'interopérer avec du code non managé et un attribut qui permet d'accéder aux membres de module sans le nom du module.

Dans cette section

ComClassAttribute, classe

HideModuleNameAttribute, classe

VBFixedArrayAttribute, classe

VBFixedStringAttribute, classe

Rubriques connexes

Attributs en Visual Basic

ComClassAttribute, classe

L'attribut **ComClassAttribute** demande au compilateur d'ajouter des métadonnées qui permettent à une classe d'être exposée comme un objet COM.

```
<System.AttributeUsage(System.AttributeTargets.Class, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class ComClassAttribute
   Inherits System.Attribute
```

Notes

Utilisez **ComClassAttribute** pour simplifier le processus d'exposition des composants COM à partir de Visual Basic. Les objets COM sont très différents des assemblys .NET Framework ; sans le **ComClassAttribute**, vous devez suivre un certain nombre d'étapes pour générer un objet COM à partir de Visual Basic. Nombre de ces opérations sont automatisées par le compilateur lorsqu'il rencontre des classes marquées avec l'attribut **ComClassAttribute**.

Remarque

Cet attribut simplifie la création d'objets COM. Pour exposer une classe en tant qu'objet COM, vous devez compiler le projet avec l'option **Inscrire pour COM Interop** sélectionnée dans la section **Générer** de la boîte de dialogue **Propriétés de la configuration**.

Remarque

Bien que vous puissiez également exposer une classe créée avec Visual Basic en tant qu'objet COM pour le code non managé à utiliser, il ne s'agit pas d'un vrai objet COM. Pour plus d'informations, consultez Interopérabilité COM dans les applications .NET Framework.

Exemple

Pour exécuter cet exemple, créez un nouveau projet de type **Bibliothèque de classes** et ajoutez le code suivant à un module de classe.

VB

```
    <ComClass(ComClass1.ClassId, ComClass1.InterfaceId, ComClass1.EventsId)> _
    Public Class ComClass1
        ' Use the Region directive to define a section named COM Guids.
#Region "COM GUIDs"
        ' These  GUIDs provide the COM identity for this class
        ' and its COM interfaces. You can generate
        ' these guids using guidgen.exe
        Public Const ClassId As String = "7666AC25-855F-4534-BC55-27BF09D49D46"
        Public Const InterfaceId As String = "54388137-8A76-491e-AA3A-853E23AC1217"
        Public Const EventsId As String = "EA329A13-16A0-478d-B41F-47583A761FF2"
#End Region

        Public Sub New()
            MyBase.New()
        End Sub

        Function AddNumbers(ByVal X As Integer, ByVal Y As Integer)
            AddNumbers = X + Y
        End Function
    End Class
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Procédure pas à pas : création d'objets COM avec Visual Basic 2005

Référence

Membres d'une classe ComClassAttribute

VBFixedArrayAttribute, classe

VBFixedStringAttribute, classe

System.Runtime.InteropServices

Concepts

Application d'attributs d'interopérabilité

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

Membres d'une classe ComClassAttribute

La classe ComClassAttribute fournit des propriétés qui identifient l'objet et indiquent si le nom occulte un autre membre.

Propriétés

ClassID, propriété	Contient un ID de classe utilisé pour identifier la classe de façon unique.
EventID, propriété	Contient un ID d'événement utilisé pour identifier l'événement de façon unique.
InterfaceID, propriété	Contient un ID d'interface utilisé pour identifier l'interface de façon unique.
InterfaceShadows, propriété	Indique qu'un nom d'interface COM généré occulte un autre membre de la classe ou de la classe de base.

Constructeur

ComClassAttribute, constructeur	Initialise une nouvelle instance de la classe **ComClassAttribute**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Procédure pas à pas : création d'objets COM avec Visual Basic 2005

Référence

VBFixedArrayAttribute, classe
VBFixedStringAttribute, classe
ComClassAttribute, classe
Shadows
System.Runtime.InteropServices

Concepts

Application d'attributs d'interopérabilité
Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

ComClassAttribute, constructeur

Initialise une nouvelle instance de la classe **ComClassAttribute**.

```
Public Sub New()
' -or-
Public Sub New( _
   ByVal _ClassID As String _
)
' -or-
Public Sub New( _
   ByVal _ClassID As String, _
   ByVal _InterfaceID As String _
)
' -or-
Public Sub New( _
   ByVal _ClassID As String, _
   ByVal _InterfaceID As String, _
   ByVal _EventId As String _
)
```

Paramètres

_ClassID

Initialise la valeur de la propriété **ClassID** utilisée pour identifier une classe de façon unique.

_InterfaceID

Initialise la valeur de la propriété **InterfaceID** utilisée pour identifier une interface de façon unique.

_EventId

Initialise la valeur de la propriété **EventID** utilisée pour identifier un événement de façon unique.

Notes

Utilisez le constructeur de la classe **ComClassAttribute** pour définir les propriétés **ClassID**, **InterfaceID** ou **EventID** lors de l'application de **ComClassAttribute** à une classe.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : ComClassAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Procédure pas à pas : création d'objets COM avec Visual Basic 2005

Référence

ComClassAttribute, classe

System.Runtime.InteropServices

Concepts

Attributs utilisés dans Visual Basic

Durée de vie d'un objet : création et destruction des objets

Autres ressources

Attributs en Visual Basic

ComClassAttribute.ClassID, propriété

Obtient un ID de classe utilisé pour identifier une classe de façon unique.

```
Public ReadOnly Property ClassID As String
```

Valeur de retour

Lecture seule. Chaîne qui peut être utilisée par le compilateur pour identifier la classe de façon unique lors de la création d'un objet COM.

Notes

Le constructeur définit cette propriété lorsque l'attribut **ComClassAttribute** est appliqué à une classe.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : ComClassAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

ComClassAttribute, classe

System.Runtime.InteropServices

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

ComClassAttribute.EventID, propriété

Obtient un ID d'événement utilisé pour identifier un événement de façon unique.

```
Public ReadOnly Property EventID As String
```

Valeur de retour

En lecture seule. Chaîne qui peut être utilisée par le compilateur pour identifier un événement de la classe de façon unique lors de la création d'un objet COM.

Notes

Le constructeur définit cette propriété lorsque l'attribut **ComClassAttribute** est appliqué à une classe.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : ComClassAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

ComClassAttribute, classe

System.Runtime.InteropServices

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

ComClassAttribute.InterfaceID, propriété

Obtient un ID d'interface utilisé pour identifier une interface de façon unique.

```
Public ReadOnly Property InterfaceID As String
```

Valeur de retour

Lecture seule. Chaîne qui peut être utilisée par le compilateur pour identifier une interface de la classe de façon unique lors de la création d'un objet COM.

Notes

Le constructeur définit cette propriété lorsque l'attribut **ComClassAttribute** est appliqué à une classe.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : ComClassAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

ComClassAttribute, classe

System.Runtime.InteropServices

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

ComClassAttribute.InterfaceShadows, propriété

Indique que le nom de l'interface COM occulte un autre membre de la classe ou de la classe de base.

```
Public Property InterfaceShadows As Boolean
```

Valeur de retour

Valeur **Boolean** indiquant que le nom de l'interface COM occulte un autre membre de la classe ou de la classe de base.

Notes

L'occultation a lieu quand un membre utilise le même nom qu'un autre membre.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : ComClassAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

ComClassAttribute, classe

System.Runtime.InteropServices

Concepts

Occultation dans Visual Basic

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

HideModuleNameAttribute, classe

Lorsqu'il s'applique à un module, l'attribut **HideModuleNameAttribute** permet aux membres du module d'être accessibles uniquement à l'aide de la qualification nécessaire pour le module.

```
' Usage
<HideModuleName> Module moduleName
' Declaration
<System.AttributeUsage(System.AttributeTargets.Class, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class HideModuleNameAttribute
   Inherits System.Attribute
```

Notes

Cet attribut permet d'exposer les membres du module sans afficher le module lui-même. Lors de l'utilisation de IntelliSense dans l'environnement de développement intégré (IDE) Visual Studio pour Visual Basic, le module ne s'affiche pas dans la liste des opérations terminées des instructions. IntelliSense affiche plutôt les membres du module.

Même lorsque le nom du module est masqué, les membres du module restent accessibles dans le module.

Si un nom de membre entre en conflit avec un autre identificateur ou avec un membre d'un autre module possédant l'attribut **HideModuleNameAttribute**, le membre ne s'affiche pas dans IntelliSense au niveau du module.

Exemple

Cet exemple indique comment utiliser l'attribut **HideModuleNameAttribute** pour ajouter un objet de base de données à **My** accessible via `My.Database`.

VB

```
Namespace My
    <HideModuleName()> Module CustomMyDatabase
        Public ReadOnly Property Database() As MyDatabase
            Get
                Return databaseValue
            End Get
        End Property
        Private ReadOnly databaseValue As MyDatabase = New MyDatabase
    End Module
End Namespace

Class MyDatabase
    ' The members of the My.Database object go here.
End Class
```

L'instruction **Namespace** doit apparaître en dehors des classes ou des modules.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : HideModuleNameAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedArrayAttribute, classe

VBFixedStringAttribute, classe

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

VBFixedArrayAttribute, classe

Indique qu'un tableau dans une structure ou une variable non locale doit être traité comme un tableau de longueur fixe.

```
<System.AttributeUsage(System.AttributeTargets.Field, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class VBFixedArrayAttribute
   Inherits System.Attribute
```

Notes

Par défaut, les tableaux Visual Basic ont une longueur variable. Cet attribut est utile en cas d'utilisation de fonctions d'entrée et de sortie de fichier Visual Basic, telles que **FileGet** et **FilePut**, et d'appels API qui nécessitent des tableaux de longueur fixe.

Exemple
VB

```
Structure Book
    <VBFixedArray(4)> Public Chapter() As Integer
End Structure

Sub WriteData()
    Dim FileNum As Integer = FreeFile()
    Dim MyBook As Book
    ReDim MyBook.Chapter(4)
    ' Add code to populate the array.
    MyBook.Chapter(0) = 1
    MyBook.Chapter(1) = 2
    MyBook.Chapter(2) = 3
    MyBook.Chapter(3) = 4
    MyBook.Chapter(4) = 5
    ' Write the array to a file.
    FileOpen(FileNum, "C:\testfile", OpenMode.Binary, _
             OpenAccess.Write, OpenShare.Default)
    FilePut(FileNum, MyBook) ' Write data.
    FileClose(FileNum)
End Sub
```

Remarque

L'attribut **VBFixedArrayAttribute** est informatif et n'alloue pas de stockage. L'objectif de cet attribut est de modifier la façon dont les tableaux figurant dans les structures et les variables non locales sont utilisés par des méthodes ou des appels API rec onnaissant l'attribut **VBFixedArrayAttribute**. N'oubliez pas que cet attribut ne convertit pas un tableau de taille variable en t ableau de taille fixe et que vous devez toujours allouer du stockage de groupe à l'aide des instructions **Dim** ou **ReDim**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres d'une classe VBFixedArrayAttribute
VBFixedStringAttribute, classe
ComClassAttribute, classe
FileGet, fonction
FilePut, fonction
FileOpen, fonction
MarshalAsAttribute

Autres ressources

Attributs en Visual Basic

Membres d'une classe VBFixedArrayAttribute

La classe VBFixedArrayAttribute fournit des propriétés qui identifient la taille et les limites du tableau.

Propriétés

Limites	Retourne les limites du tableau.
Longueur	Retourne la taille du tableau.

Constructeur

VBFixedArrayAttribute, constructeur	Initialise les limites d'un tableau de taille fixe.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : VBFixedArrayAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedStringAttribute, classe
ComClassAttribute, classe
MarshalAsAttribute

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

VBFixedArrayAttribute, constructeur

Initialise la valeur de la propriété **Bounds**.

```
Public Sub New( _
   ByVal UpperBound1 As Integer, _
   )
' -or-
Public Sub New( _
   ByVal UpperBound1 As Integer, _
   ByVal UpperBound2 As Integer, _
   )
```

Paramètres

UpperBound1

Initialise la valeur du champ supérieur, qui représente la taille de la première dimension du tableau.

UpperBound2

Initialise la valeur du champ supérieur, qui représente la taille de la seconde dimension du tableau.

Notes

Le constructeur s'exécute lorsque l'attribut **VBFixedArrayAttribute** est appliqué à un tableau.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : VBFixedArrayAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedStringAttribute, classe
ComClassAttribute, classe
MarshalAsAttribute Class

Concepts

Attributs utilisés dans Visual Basic
Durée de vie d'un objet : création et destruction des objets

Autres ressources

Attributs en Visual Basic

VBFixedArrayAttribute.Bounds, propriété

Retourne les limites du tableau.

```
Public ReadOnly Property Bounds() As Integer()
```

Valeur de retour

Contient un tableau d'entiers qui représente les limites du tableau.

Notes

Le constructeur définit cette propriété lorsque l'attribut **VBFixedArrayAttribute** est appliqué à un tableau.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : VBFixedArrayAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedStringAttribute, classe
ComClassAttribute, classe
MarshalAsAttribute

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

VBFixedArrayAttribute.Length, propriété

Retourne la taille du tableau.

```
Public ReadOnly Property Length As Integer
```

Valeur de retour

Contient un entier qui représente le nombre d'éléments dans le tableau.

Notes

Le constructeur définit cette propriété lorsque l'attribut **VBFixedArrayAttribute** est appliqué à un tableau.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : VBFixedArrayAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedStringAttribute, classe
ComClassAttribute, classe
MarshalAsAttribute Class

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

VBFixedStringAttribute, classe

Indique qu'une chaîne doit être traitée comme si elle était de longueur fixe.

```
<System.AttributeUsage(System.AttributeTargets.Field, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class VBFixedStringAttribute
   Inherits System.Attribute
```

Notes

Par défaut, les chaînes Visual Basic ont une longueur variable. Cet attribut est utile en cas d'utilisation de fonctions d'entrée et de sortie de fichier Visual Basic, telles que **FileGet** et **FilePut**, qui nécessitent des chaînes de longueur fixe.

Remarque

L'attribut **VBFixedStringAttribute** spécifie la longueur d'une chaîne en octets, et non en caractères.

Exemple
VB

```
Structure Person
    Public ID As Integer
    Public MonthlySalary As Decimal
    Public LastReviewDate As Long
    <VBFixedString(15)> Public FirstName As String
    <VBFixedString(15)> Public LastName As String
    <VBFixedString(15)> Public Title As String
    <VBFixedString(150)> Public ReviewComments As String
End Structure
```

Remarque

VBFixedStringAttribute est informatif et ne peut pas être utilisé pour convertir une chaîne de longueur variable en chaîne de longueur fixe. L'objectif de cet attribut est de modifier la façon dont les chaînes figurant dans les structures et les variables non locales sont utilisées par des méthodes ou des appels API qui reconnaissent l'attribut **VBFixedStringAttribute**, tels que les fonctions **Len** et **FilePut**. N'oubliez pas que cet attribut ne modifie pas la longueur réelle de la chaîne elle-même.

Notes du développeur sur Smart Device

Cette classe n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres d'une classe VBFixedStringAttribute
VBFixedArrayAttribute, classe
ComClassAttribute, classe
Len, fonction (Visual Basic)
FileGet, fonction
FilePut, fonction
FileOpen, fonction
StringBuilder
MarshalAsAttribute

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

Membres d'une classe VBFixedStringAttribute

La classe VBFixedStringAttribute fournit une propriété qui identifie la longueur de la chaîne.

Champs

VBFixedStringAttribute.Length	Obtient la longueur de la chaîne.

Constructeur

VBFixedStringAttribute, constructeur	Initialise la valeur du champ **SizeConst**.

Notes du développeur sur Smart Device

Cette classe n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedArrayAttribute, classe
VBFixedStringAttribute, classe
ComClassAttribute, classe
StringBuilder
MarshalAsAttribute

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

VBFixedStringAttribute, constructeur

Initialise la valeur du champ **SizeConst**.

```
Public Sub New( _
   ByVal Length As Integer _
)
```

Paramètres
Length

La longueur de la chaîne fixe.

Notes

Le constructeur s'exécute lorsque l'attribut **VBFixedStringAttribute** est appliqué à une chaîne.

Notes du développeur sur Smart Device

Ce constructeur n'est pas pris en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : VBFixedStringAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
VBFixedStringAttribute, classe
StringBuilder Class
MarshalAsAttribute Class
Concepts
Attributs utilisés dans Visual Basic
Durée de vie d'un objet : création et destruction des objets
Autres ressources
Attributs en Visual Basic

VBFixedStringAttribute.Length, propriété

Obtient la longueur de la chaîne.

```
Public ReadOnly Property Length As Integer
```

Valeur de retour

Retourne la longueur de la chaîne.

Notes

Le constructeur pour cet attribut définit cette propriété lorsque l'attribut **VBFixedStringAttribute** est appliqué à une chaîne.

Notes du développeur sur Smart Device

Cette propriété n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : VBFixedStringAttribute

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

VBFixedStringAttribute, classe

Concepts

Attributs utilisés dans Visual Basic

Autres ressources

Attributs en Visual Basic

Constantes et énumérations (Visual Basic)

Visual Basic fournit plusieurs constantes et énumérations prédéfinies pour les développeurs. Les constantes stockent des valeurs qui demeurent constantes lors de l'exécution d'une application. Les énumérations offrent un moyen pratique d'utiliser des ensembles de constantes connexes et d'associer des valeurs de constantes à des noms.

Dans cette section

AppWinStyle, énumération

Indique le style de fenêtre à utiliser pour le programme appelé lors de l'appel à la fonction **Shell**.

AudioPlayMode, énumération

Indique comment lire des sons lors de l'appel à des méthodes audio.

BuiltInRole, énumération

Indique le type de rôle à vérifier lors de l'appel à la méthode My.User.IsInRole.

CallType, énumération

Indique le type de la procédure qui est appelée lors de l'appel à la fonction **CallByName**.

CompareMethod, énumération

Indique comment comparer des chaînes lors de l'appel de fonctions de comparaison.

Constantes de compilation conditionnelle

Répertorie les constantes prédéfinies disponibles pour la compilation conditionnelle.

DateFormat, énumération

Indique comment afficher les dates lors de l'appel à la fonction **FormatDateTime**.

DateInterval, énumération

Indique comment déterminer et mettre en forme des intervalles de date lors de l'appel de fonctions liées aux dates.

DeleteDirectoryOption, énumération

Spécifie l'opération à effectuer lorsqu'un répertoire à supprimer contient des fichiers ou des répertoires.

DueDate, énumération

Indique la date d'échéance des paiements lors de l'appel à des méthodes financières.

FieldType, énumération

Indique si les champs de texte sont délimités ou à largeur fixe.

FileAttribute, énumération

Indique les attributs de fichier à utiliser lors de l'appel de fonctions d'accès aux fichiers.

FirstDayOfWeek, énumération

Indique le premier jour de la semaine à utiliser lors de l'appel de fonctions liées aux dates.

FirstWeekOfYear, énumération

Indique la première semaine de l'année à utiliser lors de l'appel de fonctions liées aux dates.

MsgBoxResult, énumération

Indique le bouton sur lequel vous avez appuyé dans un message, retourné par la fonction **MsgBox**.

MsgBoxStyle, énumération

Indique les boutons à afficher lors de l'appel à la fonction **MsgBox**.

OpenAccess, énumération

Indique comment ouvrir un fichier lors de l'appel à des fonctions d'accès aux fichiers.

OpenMode, énumération

Indique comment ouvrir un fichier lors de l'appel à des fonctions d'accès aux fichiers.

OpenShare, énumération

Indique comment ouvrir un fichier lors de l'appel à des fonctions d'accès aux fichiers.

Constantes d'impression et d'affichage

Répertorie les constantes prédéfinies disponibles lors de l'appel aux fonctions d'impression et d'affichage.

RecycleOption, énumération

Spécifie si un fichier doit être supprimé définitivement ou être placé dans la Corbeille.

SearchOption, énumération

Spécifie s'il faut effectuer la recherche dans tous les répertoires ou seulement dans les répertoires de niveau supérieur.

Tristate, énumération

Indique une valeur **Boolean** ou si la valeur par défaut doit être utilisée lors de l'appel aux fonctions liées au format des nombres.

UICancelOption, énumération

Spécifie l'action à effectuer si l'utilisateur clique sur **Annuler** pendant une opération.

UIOption, énumération

Spécifie si une boîte de dialogue de progression doit s'afficher ou non lors de la copie, la suppression ou le déplacement de fichiers ou de répertoires.

VariantType, énumération

Indique le type d'un objet variant, retourné par la fonction **VarType**.

VbStrConv, énumération

Indique le type de conversion à effectuer lors de l'appel à la fonction **StrConv**.

Rubriques connexes

Référence Visual Basic

Visual Basic

Constantes en Visual Basic

Vue d'ensemble des énumérations

AppWinStyle, énumération

Indique le style de fenêtre à utiliser pour le programme appelé lors de l'appel de la fonction **Shell**.

Notes

Lorsque vous soumettez des commandes **Shell**, vous pouvez utiliser les membres d'énumération suivants dans votre code à la place des valeurs réelles.

L'argument *Style* prend les membres d'énumération **AppWinStyle**.

Membres

Membre	Constante	Description
Hide	**vbHide**	La fenêtre est masquée et activée.
NormalFocus	**vbNormalFocus**	La fenêtre est activée et rétablie à sa taille et à sa position d'origine.
MinimizedFocus	**vbMinimizedFocus**	La fenêtre est affichée sous forme d'icône et activée.
MaximizedFocus	**vbMaximizedFocus**	La fenêtre est agrandie et activée.
NormalNoFocus	**vbNormalNoFocus**	La fenêtre est rétablie à sa taille et à sa position les plus récentes. La fenêtre active actuellement reste active.
MinimizedNoFocus	**vbMinimizedNoFocus**	La fenêtre est affichée sous forme d'icône. La fenêtre active actuellement reste active.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

Shell, fonction

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

AudioPlayMode, énumération

Indique comment lire des sons lors de l'appel à des méthodes audio.

Notes

Vous pouvez utiliser les membres de l'énumération **AudioPlayMode** pour contrôler la lecture des sons par la méthode My.Computer.Audio.Play.

Membres

Membre	Description
WaitToComplete	Entraîne la lecture du son par la méthode **My.Computer.Audio.Play**, et attend la fin de l'exécution avant que le code appelant continue.
Background	Entraîne la lecture du son en arrière-plan par la méthode **My.Computer.Audio.Play**. Le code appelant continue à s'exécuter.
BackgroundLoop	Entraîne la lecture du son en arrière-plan par la méthode **My.Computer.Audio.Play** jusqu'à ce que la méthode My.Computer.Audio.Stop soit appelée. Le code appelant continue à s'exécuter.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Computer.Audio.Play, méthode

AudioPlayMode

Concepts

Constantes et énumérations intrinsèques

BuiltInRole, énumération

Indique le type de rôle à vérifier lors de l'appel à la méthode My.User.IsInRole.

Notes

Cette énumération fournit des fonctionnalités similaires à l'énumération WindowsBuiltInRole.

Ces rôles représentent les groupes Windows locaux communs à la plupart des installations de Windows NT, Windows 2000 et Windows XP.

Membres

Membre	Description
AccountOperator	Les opérateurs de compte gèrent les comptes d'utilisateurs sur un ordinateur ou un domaine.
Administrator	Les administrateurs ont un accès total et illimité à l'ordinateur ou au domaine.
BackupOperator	Les opérateurs de sauvegarde peuvent substituer les restrictions de sécurité dans le but unique de sauvegarder ou de restaurer des fichiers.
Guest	Les invités sont plus limités que les utilisateurs.
PowerUser	Les utilisateurs avec pouvoir possèdent la plupart des autorisations administratives avec quelques restrictions. Ainsi, les utilisateurs avec pouvoirs peuvent exécuter des applications héritées (legacy), en plus des applications certifiées.
PrintOperator	Les opérateurs d'impression peuvent prendre le contrôle d'une imprimante.
Replicator	Les réplicateurs prennent en charge la réplication de fichiers dans un domaine.
SystemOperator	Les opérateurs système gèrent un ordinateur particulier.
User	Les utilisateurs ne peuvent pas effectuer des modifications accidentelles ou intentionnelles à l'échelle du système. Ainsi, ils peuvent exécuter des applications certifiées, mais non la plupart des applications héritées (legacy).

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.User.IsInRole, méthode

Microsoft.VisualBasic.ApplicationServices.BuiltInRole

Concepts

Constantes et énumérations intrinsèques

CallType, énumération

Indique le type de la procédure qui est appelée lors de l'appel à la fonction **CallByName**.

Notes

Lorsque vous appelez la fonction **CallByName**, vous pouvez utiliser les membres d'énumération **CallType** suivants dans votre code au lieu des valeurs réelles. L'argument *UseCallType* prend les membres d'énumération **CallType**.

Membres

Membre	Constante	Description
Method	**vbMethod**	Une méthode est appelée.
Get	**vbGet**	Une valeur de propriété est extraite.
Set	**vbSet**	Une valeur de propriété est déterminée.
Let	**vbLet**	Une valeur de propriété Object est déterminée.

Notes du développeur sur Smart Device

Cette énumération n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

CallByName, fonction

Constantes CallType pour les utilisateurs de Visual Basic 6.0

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

CompareMethod, énumération

Indique comment comparer des chaînes lors de l'appel de fonctions de comparaison.

Notes

Lorsque vous appelez des fonctions de comparaison, vous pouvez utiliser l'énumération **CompareMethod** dans votre code à la place des valeurs réelles. L'argument *Compare* prend les membres d'énumération **CompareMethod**.

Membres

Membre	Constante	Description
Binary	**vbBinaryCompare**	Effectue une comparaison binaire
Text	**vbTextCompare**	Effectue une comparaison de texte

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

Filter, fonction (Visual Basic)
InStr, fonction (Visual Basic)
InStrRev, fonction (Visual Basic)
Replace, fonction (Visual Basic)
Split, fonction (Visual Basic)
StrComp, fonction (Visual Basic)

Concepts

Constantes et énumérations intrinsèques
Quand utiliser une énumération

Constantes de compilation conditionnelle

La compilation conditionnelle vous permet de contrôler facilement au moment de la compilation le code à inclure dans votre programme.

Le tableau suivant répertorie les constantes prédéfinies disponibles pour la compilation conditionnelle.

Constante	Description
CONFIG	Chaîne qui correspond au paramètre actuel de la zone **Configuration de la solution active** dans le **Gestionnaire de configurations**.
DEBUG	Valeur **Boolean** qui peut être définie dans la boîte de dialogue **Propriétés du projet**. Par défaut, la configuration de débogage pour un projet définit **DEBUG**. Lorsque **DEBUG** est défini, les méthodes de la classe Debug génèrent un résultat vers la fenêtre **Sortie**. Lorsque la constante n'est pas définie, les méthodes de classe **Debug** ne sont pas compilées et aucune sortie de débogage n'est générée.
TARGET	Chaîne représentant le type de sortie pour le projet ou le paramètre de l'option de ligne de commande **/target**. Les valeurs possibles de TARGET sont : "winexe" pour une application Windows, "exe" pour une application console, "library" pour une bibliothèque de classes et "module" pour un module. L'option **/target** peut être définie dans l'environnement de développement intégré Visual Studio. Pour plus d'informations, consultez /target.
TRACE	Valeur **Boolean** qui peut être définie dans la boîte de dialogue **Propriétés du projet**. Par défaut, toutes les configurations pour un projet définissent **TRACE**. Lorsque **TRACE** est défini, les méthodes de la classe Trace génèrent un résultat vers la fenêtre **Sortie**. Lorsque la constante n'est pas définie, les méthodes de classe **Trace** ne sont pas compilées et aucune sortie **Trace** n'est générée.
VBC_VER	Nombre représentant la version de Visual Basic au format *major.minor*. Le numéro de version de Visual Basic 2005 est 8.0.
_MYTYPE	Chaîne représentant le type de projet créé. Elle contrôle quels sont les objets **My** disponibles dans le code. Pour plus d'informations, consultez Comment My dépend du type de projet.

Vous pouvez utiliser ces constantes uniquement pour la compilation conditionnelle ; ils ne peuvent pas être utilisés dans un code exécutable.

Exemple

Cet exemple utilise la constante de compilation conditionnelle **TARGET** pour déterminer s'il faut compiler certaines instructions.

VB

```
#If TARGET = "winexe" Then
        ' Insert code to be compiled for a Windows application.
#ElseIf TARGET = "exe" Then
```

```
        ' Insert code to be compiled for a console application.
#End If
```

Configuration requise

L'environnement de développement intégré Visual Studio définit les constantes de compilation conditionnelles **CONFIG**, **DEBUG**, **TRACE**et **_MYTYPE**.

Le compilateur Visual Basic définit les constantes de compilation conditionnelles **TARGET** et **VBC_VER**. Les constantes **TARGET**, **VBC_VER** et **_MYTYPE** ne sont pas disponibles dans les versions de compilateur antérieures à Visual Basic 2005.

Voir aussi

Tâches

Comment : déclarer des constantes de compilation conditionnelle

Référence

#If...Then...#Else, directives

#Const, directive

/target (Visual Basic)

Concepts

Vue d'ensemble de la compilation conditionnelle

Comment My dépend du type de projet

DateFormat, énumération

Indique comment afficher les dates lors de l'appel de la fonction **FormatDateTime**.

Notes

Lorsque vous appelez la fonction **FormatDateTime**, vous pouvez utiliser les membres d'énumération suivants dans votre code à la place des valeurs réelles.

Remarque
Vous pouvez accéder aux paramètres régionaux de votre ordinateur en double-cliquant sur l'icône **Options régionales** du **Panneau de configuration**.

Membres

Membre	Constante	Description
GeneralDate	**vbGeneralDate**	Pour les nombres réels, affiche une date et une heure. Si le nombre n'a pas de partie fractionnaire, affiche seulement une date. Si le nombre n'a pas de partie entière, affiche seulement une heure. Le mode d'affichage de la date et de l'heure est fonction des paramètres régionaux de votre ordinateur.
LongDate	**vbLongDate**	Affiche une date dans le format de date longue spécifié dans les paramètres régionaux de votre ordinateur.
ShortDate	**vbShortDate**	Affiche une date dans le format de date courte spécifié dans les paramètres régionaux de votre ordinateur.
LongTime	**vbLongTime**	Affiche une heure en utilisant le format d'heure complète spécifié dans les paramètres régionaux de votre ordinateur.
ShortTime	**vbShortTime**	Affiche une heure en utilisant le format d'heure abrégée spécifié dans les paramètres régionaux de votre ordinateur.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

FormatDateTime, fonction (Visual Basic)

Concepts

Chaînes de format DateTime personnalisées

Quand utiliser une énumération

DateInterval, énumération

Indique comment déterminer et mettre en forme des intervalles de date lors de l'appel de fonctions liées aux dates.

Notes

Lorsque vous appelez des fonctions liées aux dates, vous pouvez utiliser des membres de cette énumération dans votre code à la place des valeurs réelles.

L'énumération **DateInterval** définit des constantes employées avec des fonctions liées aux dates pour identifier la manière dont les intervalles de date sont déterminés et mis en forme. Le tableau suivant répertorie les membres de l'énumération **DateInterval**.

Membres

Membre	Description
Day	Jour du mois (1 à 31)
DayOfYear	Jour de l'année (1 à 366)
Hour	Heure (1 à 24)
Minute	Minute (1 à 60)
Month	Mois (1 à 12)
Quarter	Trimestre de l'année (1 à 4)
Second	Seconde (1 à 60)
Weekday	Jour de la semaine (1 à 7)
WeekOfYear	Semaine de l'année (1 à 53)
Year	Année

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

DateFormat, énumération
DueDate, énumération
DatePart, fonction (Visual Basic)
DateAdd, fonction (Visual Basic)
DateDiff, fonction (Visual Basic)

Concepts

Constantes et énumérations intrinsèques
Quand utiliser une énumération

DeleteDirectoryOption, énumération

Spécifie l'opération à effectuer lorsqu'un répertoire à supprimer contient des fichiers ou des répertoires.

Notes

Cette énumération est utilisée avec la méthode My.Computer.FileSystem.DeleteDirectory.

Membres

Membre	Description
DeleteAllContents	Supprime le contenu du répertoire ainsi que le répertoire. Par défaut.
ThrowIfDirectoryNonEmpty	Lève IOException si le répertoire n'est pas vide. La propriété **Data** de l'exception répertorie le ou les fichiers qui n'ont pas pu être supprimés.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : DeleteDirectoryOption

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : supprimer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.DeleteDirectory, méthode

Microsoft.VisualBasic.FileIO.DeleteDirectoryOption

DueDate, énumération

Indique la date d'échéance de paiements lors de l'appel à des méthodes financières.

Notes

Lorsque vous appelez des fonctions financières, vous pouvez utiliser les membres de cette énumération dans votre code à la place des valeurs réelles.

L'énumération **DueDate** définit des constantes utilisées pour identifier la date d'échéance de paiements.

Membres

Membre	Description
BegOfPeriod	Tombe au début de l'intervalle de date.
EndOfPeriod	Tombe à la fin de l'intervalle de date.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

DateInterval, énumération

Liste des mots clés financiers

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

FieldType, énumération

Indique si les champs de texte sont délimités ou à largeur fixe.

Notes

Cette énumération est utilisée avec l'objet TextFieldParser.

Membres

Nom du membre	Description
Delimited	Indique que les champs sont délimités.
FixedWidth	Indique que la largeur des champs est fixe.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Référence

TextFieldParser.TextFieldType, propriété

TextFieldParser

OpenTextFieldParser

FileAttribute, énumération

Indique les attributs de fichier à utiliser lors de l'appel de fonctions d'accès aux fichiers.

Notes

Lorsque vous appelez les fonctions **Dir**, **GetAttr** ou **SetAttr**, vous pouvez utiliser l'énumération **FileAttribute** dans votre code à la place des valeurs réelles.

L'argument *Attributes* prend les membres d'énumération **FileAttribute**.

Lors de l'exécution d'opérations d'E/S sur fichier, l'objet **My.Computer.FileSystem** fournit de meilleures performances et une plus grande facilité d'utilisation que les méthodes d'E/S sur fichier héritées (legacy). Pour plus d'informations, consultez My.Computer.FileSystem, objet.

Membres

Membre	Constante	Description
Normal	**vbNormal**	Normal (valeur par défaut pour **Dir** et **SetAttr**). Aucune caractéristique spéciale s'applique à ce fichier .
ReadOnly	**vbReadOnly**	En lecture seule.
Hidden	**vbHidden**	Caché.
System	**vbSystem**	Fichier système.
Volume	**vbVolume**	Étiquette du volume Cet attribut n'est pas valide lorsqu'il est utilisé avec **SetAttr**.
Directory	**vbDirectory**	Répertoire ou dossier.
Archive	**vbArchive**	Fichier modifié depuis la dernière sauvegarde.

Notes du développeur sur Smart Device

Cette énumération n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

Dir, fonction

GetAttr, fonction

SetAttr, fonction

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

FirstDayOfWeek, énumération

Indique le premier jour de la semaine à utiliser lors de l'appel de fonctions liées aux dates.

Notes

Lorsque vous appelez des fonctions liées aux dates, vous pouvez utiliser les membres d'énumération suivants dans votre code au lieu des valeurs réelles.

Certaines fonctions liées aux dates prennent un argument *DayOfWeek*, un argument *WeekOfYear* ou les deux. L'énumération **FirstDayOfWeek** spécifie les valeurs valides des arguments *DayOfWeek*, ainsi que les valeurs de retour possibles des fonctions.

L'argument *DayOfWeek* prend les membres d'énumération **FirstDayOfWeek**.

Les fonctions liées aux dates retournent également les valeurs **FirstDayOfWeek** (sauf **System**).

Membres

Membre	Constante	Description
System	**vbUseSystemDayOfWeek**	Le premier jour de la semaine spécifié dans vos paramètres système
Sunday	**vbSunday**	Dimanche (valeur par défaut)
Monday	**vbMonday**	Lundi
Tuesday	**vbTuesday**	Mardi
Wednesday	**vbWednesday**	Mercredi
Thursday	**vbThursday**	Jeudi
Friday	**vbFriday**	Vendredi
Saturday	**vbSaturday**	Samedi

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

FirstWeekOfYear, énumération
DateFormat, énumération
DateDiff, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
Format, fonction
Weekday, fonction (Visual Basic)

Concepts

Constantes et énumérations intrinsèques
Quand utiliser une énumération

FirstWeekOfYear, énumération

Indique la première semaine de l'année à utiliser lors de l'appel de fonctions liées aux dates.

Notes

Lorsque vous appelez des fonctions liées aux dates, vous pouvez utiliser les membres d'énumération suivants dans votre code au lieu des valeurs réelles.

Certaines fonctions liées aux dates prennent un argument *DayOfWeek*, un argument *WeekOfYear* ou les deux. L'énumération **FirstWeekOfYear** spécifie les valeurs valides des arguments *WeekOfYear*, ainsi que les valeurs de retour possibles des fonctions.

L'argument *WeekOfYear* prend les membres d'énumération **FirstWeekOfYear**.

Membres

Membre	Constante	Description
System	**vbUseSystem**	Jour de la semaine spécifié dans les paramètres système comme le premier jour de la semain e
Jan1	**vbFirstJan1**	Semaine comprenant le 1er janvier (valeur par défaut)
FirstFourDay s	**vbFirstFourDay s**	Première semaine comportant au moins quatre jours dans la nouvelle année
FirstFullWee k	**vbFirstFullWee k**	Première semaine complète de l'année

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

FirstDayOfWeek, énumération
DateFormat, énumération
DateDiff, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
Format, fonction

Concepts

Constantes et énumérations intrinsèques
Quand utiliser une énumération

MsgBoxResult, énumération

Indique le bouton sur lequel vous avez appuyé dans un message, retourné par la fonction **MsgBox**.

Notes

Lorsque vous appelez la fonction **MsgBox**, vous pouvez utiliser l'énumération **MsgBoxResult** dans votre code à la place des valeurs réelles.

La fonction **MsgBox** retourne une valeur d'énumération **MsgBoxResult**.

Membres

Membre	Constante	Description
OK	**vbOK**	L'utilisateur a cliqué sur **OK**.
Cancel	**vbCancel**	L'utilisateur a cliqué sur **Annuler**.
Abort	**vbAbort**	L'utilisateur a cliqué sur **Abandonner**.
Retry	**vbRetry**	L'utilisateur a cliqué sur **Réessayer**.
Ignore	**vbIgnore**	L'utilisateur a cliqué sur **Ignorer**.
Yes	**vbYes**	L'utilisateur a cliqué sur **Oui**.
No	**vbNo**	L'utilisateur a cliqué sur **Non**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

MsgBoxStyle, énumération

MsgBox, fonction (Visual Basic)

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

MsgBoxStyle, énumération

Indique les boutons à afficher lors de l'appel à la fonction **MsgBox**.

Notes

Lorsque vous appelez la fonction **MsgBox**, vous pouvez utiliser l'énumération **MsgBoxStyle** dans votre code à la place des valeurs réelles. L'argument *Buttons* prend les membres d'énumération **MsgBoxStyle**.

Membres

Membre	Constante	Description
OKOnly	**vbOKOnly**	Bouton **OK** uniquement (valeur par défaut)
OKCancel	**vbOKCancel**	Boutons **OK** et **Annuler**
AbortRetryIgnore	**vbAbortRetryIgnore**	Boutons **Abandonner**, **Réessayer** et **Ignorer**
YesNoCancel	**vbYesNoCancel**	Boutons **Oui**, **Non** et **Annuler**
YesNo	**vbYesNo**	Boutons **Oui** et **Non**
RetryCancel	**vbRetryCancel**	Boutons **Réessayer** et **Annuler**
Critical	**vbCritical**	Message critique
Question	**vbQuestion**	Requête d'avertissement
Exclamation	**vbExclamation**	Message d'avertissement
Information	**vbInformation**	Message d'information
DefaultButton1	**vbDefaultButton1**	Le premier bouton est le bouton par défaut.
DefaultButton2	**vbDefaultButton2**	Le deuxième bouton est le bouton par défaut.
DefaultButton3	**vbDefaultButton3**	Le troisième bouton est le bouton par défaut.
ApplicationModal	**vbApplicationModal**	Boîte de dialogue modale d'application
SystemModal	**vbSystemModal**	Boîte de dialogue modale système
MsgBoxSetForeground	**vbMsgBoxSetForeground**	Fenêtre du message de premier plan
MsgBoxRight	**vbMsgBoxRight**	Alignement à droite du texte
MsgBoxRtlReading	**vbMsgBoxRtlReading**	Lecture de texte de droite à gauche (systèmes hébreu et arabe)
MsgBoxHelp	**vbMsgBoxHelp**	Texte d'aide

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

MsgBoxResult, énumération

MsgBox, fonction (Visual Basic)

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

OpenAccess, énumération

Indique comment ouvrir un fichier lors de l'appel à des fonctions d'accès aux fichiers.

Notes

Lorsque vous appelez des fonctions relatives à l'accès aux fichiers, vous pouvez utiliser des membres de cette énumération dans votre code à la place des valeurs réelles.

L'énumération **OpenAccess** définit les constantes utilisées pour déterminer si un fichier peut être lu (**Read**), écrit (**Write**) ou les deux (**ReadWrite**). Un fichier ouvert pour un accès **Read** ne peut pas être modifié ; il peut être uniquement lu. Le tableau suivant répertorie les membres de l'énumération **OpenAccess**.

Lors de l'exécution d'opérations d'E/S sur fichier, l'objet **My.Computer.FileSystem** fournit de meilleures performances et une plus grande facilité d'utilisation que les méthodes d'E/S sur fichier héritées (legacy). Pour plus d'informations, consultez My.Computer.FileSystem, objet.

Membres

Membre	Description
Default	Accès en lecture et en écriture autorisé. Il s'agit de l'option par défaut.
Read	Accès en lecture autorisé.
ReadWrite	Accès en lecture et en écriture autorisé.
Write	Accès en écriture autorisé.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

OpenMode, énumération

OpenShare, énumération

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

OpenMode, énumération

Indique comment ouvrir un fichier lors de l'appel à des fonctions d'accès aux fichiers.

Notes

Lorsque vous appelez des fonctions relatives à l'accès aux fichiers, vous pouvez utiliser des membres de cette énumération dans votre code à la place des valeurs réelles.

L'énumération **OpenMode** définit les constantes utilisées pour déterminer les modes d'accès aux fichiers. Le tableau suivant répertorie les membres de l'énumération **OpenMode**.

Input, **Output** et **Append** sont utilisés pour accéder séquentiellement à des fichiers, tels que des fichiers texte, tandis que **Binary** est utilisé pour un accès binaire et **Random** pour un accès aléatoire.

Lorsque vous accédez séquentiellement à un fichier, vous ne pouvez pas modifier ses données. Vous pouvez lire les données, en ajouter ou les remplacer par de nouvelles. Si vous ouvrez le fichier pour l'entrée, le contenu du fichier sera remplacé, même si vous n'écrivez pas directement dans le fichier.

Lors de l'exécution d'opérations d'E/S sur fichier, l'objet **My.Computer.FileSystem** fournit de meilleures performances et une plus grande facilité d'utilisation que les méthodes d'E/S sur fichier héritées (legacy). Pour plus d'informations, consultez My.Computer.FileSystem, objet.

Membres

Membre	Description
Append	Fichier ouvert pour être ajouté. Par défaut.
Binary	Fichier ouvert pour un accès binaire.
Input	Fichier ouvert pour un accès en lecture.
Output	Fichier ouvert pour un accès en écriture.
Random	Fichier ouvert pour un accès aléatoire.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

OpenShare, énumération

OpenAccess, énumération

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

OpenShare, énumération

Indique comment ouvrir un fichier lors de l'appel à des fonctions d'accès aux fichiers.

Notes

Lorsque vous appelez des fonctions relatives à l'accès aux fichiers, vous pouvez utiliser les membres de cette énumération dans votre code à la place des valeurs réelles.

L'énumération **OpenShare** définit des constantes qui spécifient si d'autres processus peuvent accéder ou non au fichier lorsque celui-ci est ouvert dans votre application. Le tableau suivant répertorie les membres de l'énumération **OpenShare**.

Lors de l'exécution d'opérations d'E/S sur fichier, l'objet **My.Computer.FileSystem** fournit de meilleures performances et une plus grande facilité d'utilisation que les méthodes d'E/S sur fichier héritées (legacy). Pour plus d'informations, consultez My.Computer.FileSystem, objet.

Membres

Membre	Description
Default	**LockReadWrite**. Il s'agit de la valeur par défaut.
LockRead	Les autres processus ne peuvent pas lire le fichier.
LockReadWrite	Les autres processus ne peuvent pas lire le fichier ou écrire dedans.
LockWrite	Les autres processus ne peuvent pas écrire dans le fichier.
Shared	N'importe quel processus peut lire le fichier ou écrire dedans.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

OpenAccess, énumération

OpenMode, énumération

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

Constantes d'impression et d'affichage

Lorsque vous appelez des fonctions d'impression et d'affichage, vous pouvez utiliser les constantes suivantes dans votre code au lieu des valeurs réelles.

Membres du module ControlChars

Membre	Constante	Équivalent	Description
CrLf	**vbCrLf**	**Chr(**13**) + Chr(**10**)**	Combinaison de retour chariot et de saut de ligne
Cr	**vbCr**	**Chr(**13**)**	Caractère de retour chariot
Lf	**vbLf**	**Chr(**10**)**	Caractère de saut de ligne
NewLine	**vbNewLine**	**Chr(**13**) + Chr(**10**)**	Caractère de saut de ligne
NullChar	**vbNullChar**	**Chr(**0**)**	Caractère ayant la valeur 0
N/A	**vbNullString**	Chaîne ayant la valeur 0	Différent d'une chaîne de longueur nulle ("") ; utilisé pour l'appel de procédures externes
N/A	**vbObjectError**	-2147221504	Numéro de l'erreur. Les numéros d'erreur définis par l'utilisateur doivent être supérieurs à cette valeur. Par exemple : `Err.Raise(Number) = vbObjectError + 1000`
Tab	**vbTab**	**Chr(**9**)**	Caractère de tabulation
Back	**vbBack**	**Chr(**8**)**	Caractère de retour arrière
FormFeed	**vbFormFeed**	**Chr(**12**)**	Non utilisé dans Microsoft Windows
VerticalTab	**vbVerticalTab**	**Chr(**11**)**	Non utilisé dans Microsoft Windows
Quote	N/A	**Chr(**34**)**	Caractère de guillemet (" ou ') utilisé pour encadrer des valeurs.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Constants

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Fonctions Print, PrintLine

Fonctions Write, WriteLine

Concepts

Constantes et énumérations intrinsèques

Vue d'ensemble des constantes

Constantes déclarées par Visual Basic

RecycleOption, énumération

Spécifie si un fichier doit être supprimé définitivement ou être placé dans la **Corbeille**.

Notes

Cette énumération est utilisée avec les méthodes My.Computer.FileSystem.DeleteDirectory et My.Computer.FileSystem.DeleteFile.

Pour éviter que les utilisateurs ne suppriment accidentellement des fichiers importants, la **Corbeille** collecte les fichiers supprimés et permet de les restaurer si nécessaire. Vous devez supprimer explicitement les fichiers de la **Corbeille** pour rendre la suppression permanente. Tant que les fichiers ne sont pas supprimés de la **Corbeille**, l'espace disque occupé par ces fichiers n'est pas disponible pour d'autres utilisations.

Il peut parfois s'avérer bénéfique de contourner la **Corbeille**, surtout si vous supprimez un nombre important de fichiers (ou un nombre plus petit de fichiers volumineux) dont vous êtes certain de ne plus avoir besoin, et que vous souhaitez récupérer l'espace disque qu'ils occupaient.

Membres

Membre	Description
DeletePermanently	Supprime définitivement le fichier ou le répertoire. Par défaut.
SendToRecycleBin	Envoie le fichier ou le répertoire à la **Corbeille**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : RecycleOption

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : supprimer un fichier dans Visual Basic

Comment : supprimer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.DeleteDirectory, méthode

My.Computer.FileSystem.DeleteFile, méthode

Microsoft.VisualBasic.FileIO.RecycleOption

SearchOption, énumération

Spécifie s'il faut effectuer la recherche dans tous les répertoires ou seulement dans les répertoires de niveau supérieur.

Notes

Cette énumération est utilisée avec les méthodes My.Computer.FileSystem.GetFiles, My.Computer.FileSystem.GetDirectories et My.Computer.FileSystem.FindInFiles.

Membres

Membre	Description
SearchAllSubDirectories	Effectue la recherche dans le répertoire spécifié et tous les sous-répertoires qu'il contient. Par défaut.
SearchTopLevelOnly	Effectue la recherche uniquement dans le répertoire spécifié en excluant les sous-répertoires.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : SearchOption

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Computer.FileSystem.FindInFiles, méthode
My.Computer.FileSystem.GetDirectories, méthode
My.Computer.FileSystem.GetFiles, méthode
Microsoft.VisualBasic.FileIO.SearchOption

Tristate, énumération

Indique une valeur Boolean ou si la valeur par défaut doit être utilisée lors de l'appel de fonctions liées au format des nombres.

Notes

Lorsque vous appelez les fonctions liées au format des nombres, vous pouvez utiliser les membres d'énumération suivants dans votre code à la place des valeurs réelles.

Membres

Membre	Description
True	Vrai. La valeur numérique de ce membre est -1.
False	Faux. La valeur numérique de ce membre est 0.
UseDefault	Paramètre par défaut. La valeur numérique de ce membre est -2.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

FormatNumber, fonction (Visual Basic)
FormatCurrency, fonction (Visual Basic)
FormatPercent, fonction (Visual Basic)

Concepts

Quand utiliser une énumération

UICancelOption, énumération

Spécifie si une exception est levée lorsque l'utilisateur clique sur **Annuler** pendant une opération.

Notes

Cette énumération détermine si une IOException est levée lorsqu'un utilisateur clique sur **Annuler** pendant une opération. Dans les deux cas, l'opération est annulée.

Membres

Nom du membre	Description
DoNothing	Ne fait rien lorsque l'utilisateur clique sur **Annuler**.
ThrowException	Lève une exception lorsque l'utilisateur clique sur **Annuler**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : UICancelOption

Plates-formes : Windows 95, Windows 98, Windows 98 Deuxième Édition, Windows Millennium, Windows NT 4.0, Windows NT Workstation 4.0, Windows NT Server 4.0, Windows 2000, Windows 2000 Server, Windows 2000 Advanced Server, Windows XP Édition familiale, Windows XP Professionnel, Windows Server 2003, .NET Compact Framework - Téléphone mobile, .NET Compact Framework, norme de l'infrastructure du langage commun (CLI, Common Language Infrastructure)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Computer.FileSystem.CopyDirectory, méthode
My.Computer.FileSystem.CopyFile, méthode
My.Computer.FileSystem.DeleteDirectory, méthode
My.Computer.FileSystem.DeleteFile, méthode
My.Computer.Network.DownloadFile, méthode
My.Computer.FileSystem.MoveDirectory, méthode
My.Computer.FileSystem.MoveFile, méthode
My.Computer.Network.UploadFile, méthode
UICancelOption

UIOption, énumération

Spécifie les boîtes de dialogue à afficher lors de la copie, de la suppression ou du déplacement de fichiers ou de répertoires.

Notes

Cette énumération est utilisée avec les méthodes My.Computer.FileSystem.CopyDirectory, My.Computer.FileSystem.CopyFile, My.Computer.FileSystem.DeleteDirectory, My.Computer.FileSystem.DeleteFile, My.Computer.FileSystem.MoveDirectory et My.Computer.FileSystem.MoveFile.

Membres

Membre	Description
OnlyErrorDialogs	Affiche uniquement les boîtes de dialogue d'erreur et masque les boîtes de dialogue de progression. Par défaut.
AllDialogs	Affiche la boîte de dialogue de progression ainsi que les boîtes de dialogue d'erreur.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : UIOption

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Computer.FileSystem.CopyDirectory, méthode
My.Computer.FileSystem.CopyFile, méthode
My.Computer.FileSystem.DeleteDirectory, méthode
My.Computer.FileSystem.DeleteFile, méthode
My.Computer.FileSystem.MoveDirectory, méthode
My.Computer.FileSystem.MoveFile, méthode
Microsoft.VisualBasic.FileIO.UIOption

VariantType, énumération

Indique le type d'un objet variant, retourné par la fonction **VarType**.

Notes

Lorsque vous appelez la fonction **VarType**, vous pouvez utiliser des membres de cette énumération dans votre code à la place des valeurs réelles.

L'énumération **VariantType** définit les constantes utilisées pour identifier les types **Variant** possibles. Le tableau suivant répertorie les membres de l'énumération **VariantType**.

Membres

Membre	Constante	Description
Array	**vbArray**	Tableau
Boolean	**vbBoolean**	**Boolean**. (**True** ou **False**.)
Byte	**vbByte**	**Byte**. (0 à 255.)
Char	**vbChar**	**Char**. (0 à 65 535.)
Currency	**vbCurrency**	Monétaire
DataObject		DataObject
Date	**vbDate**	**Date**. (0:00:00 le 1er janvier 0001 à 23:59:59 le 31 décembre 9999.)
Decimal	**vbDecimal**	**Decimal**. (0 à +/-79 228 162 514 264 337 593 543 950 335 sans décimale ; 0 à +/-7,9228162514264337593543950335 avec 28 décimales ; le plus petit nombre différent de zéro étant +/-0,0000000000000000000000000001.)
Double	**vbDouble**	**Double**. (-1,79769313486231E+308 à -4,94065645841247E-324 pour les valeurs négatives ; 4,94065645841247E-324 à 1,79769313486231E+308 pour les valeurs positives.)
Empty	**vbEmpty**	Référence null
Error		System.Exception
Integer	**vbInteger**	**Integer**. (-2 147 483 648 à 2 147 483 647.)
Long	**vbLong**	**Long**. (-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807.)
Null	**vbNull**	Objet null
Object	**vbObject**	N'importe quel type peut être stocké dans une variable de type **Object**.
Short		**Short**. (-32 768 à 32 767.)

Single	**vbSingle**	**Single**. (-3,402823E+38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E+38 pour les valeurs positives.)
String	**vbString**	**String**. (0 à environ 2 milliards de caractères Unicode.)
UserDefinedType	**vbUserDefinedType**	Type défini par l'utilisateur. Chaque membre de la structure présente une plage déterminée par son type de données et qui est indépendante des plages des autres membres.
Variant	**vbVariant**	**Variant**.

Notes du développeur sur Smart Device

Cette énumération n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

VarType, fonction (Visual Basic)

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

VbStrConv, énumération

Indique le type de conversion à effectuer lors de l'appel à la fonction **StrConv**.

Notes

Lorsque vous appelez la fonction **StrConv**, vous pouvez utiliser les membres d'énumération suivants dans votre code à la place des valeurs réelles. L'argument *Conversion* prend les membres d'énumération **VbStrConv**.

Membres

Membre	Constante	Description
UpperCase	**vbUpperCase**	Convertit la chaîne en majuscules.
LowerCase	**vbLowerCase**	Convertit la chaîne en minuscules.
ProperCase	**vbProperCase**	Convertit en majuscules la première lettre de chaque mot d'une chaîne.
Wide	**vbWide**	Convertit les caractères étroits (codés sur un octet) d'une chaîne en caractères larges (codés sur deux octets). S'applique aux paramètres régionaux asiatiques.
Narrow	**vbNarrow**	Convertit les caractères larges (codés sur deux octets) d'une chaîne en caractères étroits (codés sur un octet). S'applique aux paramètres régionaux asiatiques.
None		N'effectue pas de conversion.
Katakana	**vbKatakana**	Convertit les caractères Hiragana de la chaîne en caractères Katakana. S'applique uniquement aux paramètres régionaux japonais.
Hiragana	**vbHiragana**	Convertit les caractères Katakana de la chaîne en caractères Hiragana. S'applique uniquement aux paramètres régionaux japonais.
SimplifiedChinese	**vbSimplifiedChinese**	Convertit la chaîne en caractères chinois simplifiés.
TraditionalChinese	**vbTraditionalChinese**	Convertit la chaîne en caractères chinois traditionnels.
LinguisticCasing	**vbLinguisticCasing**	Convertit la chaîne des règles du système de fichiers pour correspondre à la casse des règles linguistiques.

Notes du développeur sur Smart Device

Cette énumération n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : faire référence à un membre d'énumération

Référence

StrConv, fonction

Concepts

Constantes et énumérations intrinsèques

Quand utiliser une énumération

Autres ressources

Introduction aux chaînes dans Visual Basic

Liste des types de données (Visual Basic)

Le tableau suivant illustre les types de données Visual Basic, leurs types Common Language Runtime pris en charge, leur allocation de stockage nominal et leur plage de valeurs.

Type Visual Basic	Structure de type Common Language Runtime	Allocation de stockage nominal	Plage de valeurs
Boolean	Boolean	Dépend de la plate-forme d'implémentation	**True** ou **False**
Byte	Byte	1 octet	0 à 255 (non signé)
Char (caractère unique)	Char	2 octets	0 à 65 535 (non signé)
Date	DateTime	8 octets	Du 1er janvier 0001 0:00:00 (minuit) au 31 décembre 9999 23:59:59
Decimal	Decimal	16 octets	0 à +/-79 228 162 514 264 337 593 543 950 335 (+/-7,9...E+28) † sans virgule décimale ; 0 à +/-7,9228162514264337593543950335 avec 28 décimales après la virgule ; la plus petite valeur différente de zéro est +/-0,0000000000000000000000000001 (+/-1E-28) †
Double (nombre à virgule flottante double précision)	Double	8 octets	-1,79769313486231570E+308 à -4,94065645841246544E-324 † pour les valeurs négatives ; 4,94065645841246544E-324 à 1,79769313486231570E+308 † pour les valeurs positives
Integer	Int32	4 octets	-2 147 483 648 à 2 147 483 647 (signé)
Long (entier long)	Int64	8 octets	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 (9,2...E+18 †) (signé)
Object	Object (classe)	4 octets sur une plate-forme 32 bits 8 octets sur une plate-forme 64 bits	N'importe quel type peut être stocké dans une variable de type **Object**
SByte	SByte	1 octet	-128 à 127 (signé)
Short (entier court)	Int16	2 octets	-32 768 à 32 767 (signé)
Single (nombre à virgule flottante simple précision)	Single	4 octets	-3,4028235E+38 à -1,401298E-45 † pour les valeurs négatives ; 1,401298E-45 à 3,4028235E+38 † pour les valeurs positives

String (longueur variable)	String (classe)	Dépend de la plate-forme d'implémentation	0 à environ 2 milliards de caractères Unicode
UInteger	UInt32	4 octets	0 à 4 294 967 295 (non signé)
ULong	UInt64	8 octets	0 à 18 446 744 073 709 551 615 (1,8...E+19 †) (non signé)
User-Defined (structure)	(hérite de ValueType)	Dépend de la plate-forme d'implémentation	Chaque membre de la structure présente une plage déterminée par son type de données et qui est indépendante des plages des autres membres
UShort	UInt16	2 octets	0 à 65 535 (non signé)

† En *notation scientifique*, "E" fait référence à une puissance de 10. Ainsi, 3,56E+2 signifie *3.56 x* 10^2 ou 356 et 3,56E-2 signifie *3.56 /* 10^2 ou 0,0356.

Remarque

Pour les chaînes contenant du texte, utilisez la fonction StrConv pour effectuer une conversion d'un format de texte à un autre.

Consommation de mémoire

Lorsque vous déclarez un type de données élémentaire, il est risqué de supposer que sa consommation de mémoire est identique à son allocation de stockage nominal. Cela est dû aux considérations suivantes :

- **Assignation de stockage.** Le Common Language Runtime peut assigner le stockage en fonction des caractéristiques actuelles de la plate-forme sur laquelle s'exécute votre application. Si la mémoire est presque pleine, il se peut qu'il compresse vos éléments déclarés aussi étroitement que possible. Dans d'autres cas, il se peut qu'il aligne leurs adresses mémoire sur les limites matérielles naturelles pour optimiser les performances.
- **Largeur de plate-forme.** L'assignation de stockage est différente sur une plate-forme 64 bits et sur une plate-forme 32 bits.

Types de données composites

Les mêmes considérations s'appliquent à chaque membre d'un type de données composite (par exemple, une structure ou un tableau). Il ne suffit pas d'additionner les allocations de stockage nominal des membres du type. D'autre part, il existe d'autres considérations, telles que les suivantes :

- **Charge mémoire.** La configuration requise pour la mémoire est plus importante pour certains types composites. Par exemple, un tableau utilise davantage de mémoire pour le tableau lui-même, mais également pour chaque dimension. Sur une plate-forme 32 bits, cette charge mémoire est actuellement de 12 octets, auxquels il convient d'ajouter 8 octets pour chaque dimension. Sur une plate-forme 64 bits, cette configuration requise est doublée.
- **Disposition de stockage.** Il est risqué de supposer que l'ordre de stockage dans la mémoire est le même que l'ordre de vos déclarations. Vous ne pouvez même pas faire de suppositions au sujet de l'alignement des octets, tels qu'une limite de 2 octets ou de 4 octets. Si vous définissez une classe ou une structure et que vous devez contrôler la disposition de stockage de ses membres, vous pouvez appliquer l'attribut StructLayoutAttribute à la classe ou à la structure.

Charge mémoire d'un objet

Un **Object** faisant référence à un type de données élémentaire ou composite utilise 4 octets en plus des données contenues dans le type de données.

Voir aussi

Référence

Fonctions de conversion de types de données
Liste des conversions
StrConv, fonction
StructLayoutAttribute Class

Concepts

Caractères de type
Utilisation efficace des types de données

Boolean, type de données (Visual Basic)

Stocke des valeurs qui peuvent être uniquement **True** ou **False**. Les mots clé **True** et **False** correspondent aux deux états des variables **Boolean**.

Notes

Utilisez le type de données **Boolean** pour contenir des valeurs à deux états, tels que true/false, oui/non ou actif/inactif.

La valeur par défaut de **Boolean** est **False**.

Conversions de type

Lorsque Visual Basic convertit des valeurs de type de données numériques en **Boolean**, 0 devient **False** et toutes les autres valeurs deviennent **True**. Lorsque Visual Basic convertit des valeurs **Boolean** en types numériques, **False** devient 0 et **True** devient -1.

Lorsque vous convertissez des valeurs **Boolean** et des types de données numériques, n'oubliez pas que les méthodes de conversion .NET Framework ne produisent pas toujours le même résultat que les mots clé de conversion Visual Basic. En effet, la conversion Visual Basic conserve un comportement compatible avec les versions antérieures. Pour plus d'informations, consultez Dépannage des types de données.

Conseils de programmation

- **Nombres négatifs. Boolean** n'est pas un type numérique et ne peut pas représenter de valeur négative. Dans tous les cas, vous ne devez pas utiliser **Boolean** pour stocker des valeurs numériques.
- **Caractères de type. Boolean** n'a aucun caractère de type de littéral ou caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Boolean.

Exemple

Dans l'exemple suivant, `runningVB` est une variable de type **Boolean** qui enregistre une simple valeur yes/no.

```
Dim runningVB As Boolean
' Check to see if program is running on Visual Basic engine.
If scriptEngine = "VB" Then
    runningVB = True
End If
```

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
CType, fonction
System.Boolean

Concepts

Utilisation efficace des types de données

Byte, type de données (Visual Basic)

Contient des entiers 8 bits (1 octet) non signés dont la valeur est comprise entre 0 et 255.

Notes

Utilisez le type de données **Byte** pour contenir les données binaires.

La valeur par défaut de **Byte** est 0.

Conseils de programmation

- **Nombres négatifs.** Dans la mesure où **Byte** est un type non signé, il ne peut représenter un nombre négatif. Si vous utilisez l'opérateur moins unaire (**-**) dans une expression qui correspond au type **Byte**, Visual Basic convertit d'abord l'expression en **Short**.
- **Conversions de format.** Lorsque Visual Basic lit ou écrit des fichiers, ou lorsqu'il appelle des DLL, des méthodes et des propriétés, il peut convertir automatiquement entre différents formats de données. Les données binaires stockées dans des variables et des tableaux **Byte** sont préservées durant les conversions de format. Vous ne devez pas utiliser une variable de type **String** pour des données binaires ; en effet, son contenu peut être endommagé pendant la conversion du format ANSI au format Unicode.
- **Extension.** Le type de données **Byte** s'étend à **Short**, **UShort**, **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single** ou **Double**. Ceci signifie que vous pouvez convertir **Byte** en ces types sans rencontrer d'erreur System.OverflowException.
- **Caractères de type. Byte** n'a aucun caractère de type de littéral ou caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Byte.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
System.Byte

Concepts

Utilisation efficace des types de données

Char, type de données (Visual Basic)

Contient des points de code 16 bits (2 octets) non signés dont la valeur est comprise entre 0 et 65 535. Chaque *point de code*, ou code de caractère, représente un seul caractère Unicode.

Notes

Utilisez le type de données **Char** lorsque vous devez contenir un caractère unique et n'avez pas besoin de la charge de **String**. Dans certains cas, vous pouvez utiliser **Char()**, un tableau d'éléments **Char**, pour contenir plusieurs caractères.

La valeur par défaut de **Char** est le caractère avec un point de code de 0.

Caractères Unicode

Les premiers 128 points de code (0–127) Unicode correspondent aux lettres et aux symboles du clavier américain standard. Ce sont les mêmes que ceux définis par le jeu de caractères ASCII. Les 128 points de code suivants (128–255) représentent les caractères spéciaux, tels que les lettres de l'alphabet latin, les accents, les symboles monétaires et les fractions. Unicode utilise les points de code restants (256-65 535) pour une large gamme de symboles, y compris les caractères textuels mondiaux, les signes diacritiques, ainsi que les symboles mathématiques et techniques.

Vous pouvez utiliser des méthodes telles que IsDigit et IsPunctuation sur une variable **Char** pour déterminer sa classification Unicode.

Conversions de type

Visual Basic n'opère pas de conversion directe entre le type **Char** et les types numériques. Vous pouvez utiliser Fonctions Asc, AscW pour convertir une valeur **Char** en un **Integer** représentant son point de code. Vous pouvez utiliser Fonctions Chr, ChrW pour convertir une valeur **Integer** en **Char** représentant son point de code.

Si le commutateur de vérification de type (instruction Option Strict) est activé, vous devez ajouter le caractère de type de littéral à un littéral de chaîne à un caractère pour l'identifier comme type de données **Char**. L'exemple suivant illustre ce comportement :

```
Option Strict On
Dim charVar As Char
' The following statement attempts to convert a String literal to Char.
' Because Option Strict is On, it generates a compiler error.
charVar = "Z"
' The following statement succeeds because it specifies a Char literal.
charVar = "Z"C
```

Conseils de programmation

- **Nombres négatifs. Char** est un type non signé et ne peut pas représenter de valeur négative. Dans tous les cas, vous ne devez pas utiliser **Char** pour stocker des valeurs numériques.
- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, par exemple des objets Automation ou COM, n'oubliez pas que les types de caractères possèdent une largeur des données différente (8 bits) dans d'autres environnements. Si vous passez un argument de 8 bits à un tel composant, déclarez-le comme type de données **Byte** et non comme **Char** dans votre nouveau code Visual Basic.
- **Extension.** Le type de données **Char** s'étend à **String**. Cela signifie que vous pouvez convertir **Char** en **String** sans produire d'erreur System.OverflowException.
- **Caractères de type.** L'ajout de caractères de type de littéral **C** à un littéral de chaîne à un caractère force ce dernier en type de données **Char**. **Char** ne possède pas de caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Char.

Voir aussi

Tâches

Comment : optimiser le stockage d'entiers positifs avec les types non signés

Comment : appeler une fonction Windows qui possède des types non signés

Référence

Liste des types de données (Visual Basic)

String, type de données (Visual Basic)

Fonctions de conversion de types de données
Liste des conversions
Fonctions Asc, AscW
Fonctions Chr, ChrW
System.Char

Concepts

Utilisation efficace des types de données

Type de données date (Visual Basic)

Contient des valeurs sous la forme IEEE 64 bits (8 octets) qui représentent des dates comprises entre le 1er janvier de l'année 1 et le 31 décembre 9999 et des heures comprises entre 0:00:00 (minuit) et 23:59:59.9999999. Chaque incrément représente 100 nanosecondes du temps écoulé depuis le 1er janvier de l'an 1 du calendrier grégorien. La valeur maximale représente 100 nanosecondes avant le début du 1er janvier de l'année 10 000.

Notes

Utilisez le type de données **Date** pour contenir des valeurs de date, des valeurs d'heure ou des valeurs d'horodatage.

La valeur par défaut de **Date** est 0:00:00 (minuit) du 1er janvier 0001.

Pré-requis de format

Vous devez joindre un littéral **Date** entre des signes dièse (**# #**). Vous devez spécifier la valeur de date au format M/d/yyyy, par exemple `#5/31/1993#`. Cette configuration requise est indépendante de vos paramètres régionaux et de vos paramètres de format de date et d'heure.

Cette restriction est due au fait que la signification de votre code ne doit jamais changer en fonction de vos paramètres régionaux avec lesquels votre application est en cours d'exécution. Si vous codez en dur un littéral **Date** de `#3/4/1998#` afin qu'il corresponde au 4 mars 1998, cette date sera compilée comme vous le souhaitez dans les paramètres régionaux qui utilisent mm/jj/aaaa. Mais si vous déployez votre application dans plusieurs pays. Dans les paramètres locaux qui utilisent le format jj/mm/aaaa, votre littéral codé en dur compilera sous la forme "3 avril 1998". Dans les paramètres régionaux qui utilisent le format aaaa/mm/jj, le littéral ne sera pas valide (avril 1998, 0003) et entraînera une erreur du compilateur.

Solutions

Pour convertir un littéral **Date** au format de vos paramètres régionaux ou à un format personnalisé, fournissez le littéral à la fonction Format, en spécifiant les formats de date/heure prédéfinis (fonction Format) ou les formats de date/heure définis par l'utilisateur (fonction Format). C'est ce qu'illustre l'exemple suivant.

```
MsgBox("The formatted date is " & Format(#5/31/1993#, "dddd, d MMM yyyy"))
```

Sinon, vous pouvez utiliser un des constructeurs surchargés de la structure DateTime pour assembler une valeur d'horodatage. L'exemple suivant crée une valeur pour représenter le 31 mai, 1993 à 12:14.

```
Dim dateInMay As New System.DateTime(1993, 5, 31, 12, 14, 0)
```

Format d'heure

Vous pouvez spécifier la valeur d'heure au format 12 heures ou 24 heures, par exemple `#1:15:30 PM#` ou `#13:15:30#`. Toutefois, si vous ne spécifiez pas les minutes, ni les secondes, vous devez indiquer s'il s'agit du matin ou de l'après-midi (AM/PM).

Valeurs par défaut de date et d'heure

Si vous n'incluez pas de date dans un littéral de date et d'heure, Visual Basic fixe la partie date de la valeur au 1er janvier 0001. Si vous n'incluez pas d'heure dans un littéral de date et d'heure, Visual Basic fixe la partie heure à minuit (00:00:00).

Conversions de type

Si vous convertissez une valeur de type **Date** en type **String**, Visual Basic restitue la date en se basant sur le format de date courte spécifié par les paramètres régionaux d'exécution, et restitue l'heure en se basant sur le format d'heure (12 heures ou 24 heures) spécifié par les paramètres régionaux d'exécution.

Conseils de programmation

- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, tels que des objets Automation ou COM, gardez à l'esprit que les types date/heure d'autres environnements ne sont pas compatibles avec le type **Date** Visual Basic. Si vous passez un argument date/heure à un tel composant, déclarez-le comme type de données **Double** et non comme **Date** dans votre nouveau code Visual Basic, et utilisez les méthodes de conversion System.DateTime.FromOADate(System.Double) et System.DateTime.ToOADate.
- **Caractères de type. Date** n'a aucun caractère de type de littéral ou caractère de type d'identificateur. Toutefois, le compilateur assimile les littéraux compris entre signes dièse (**# #**) en tant que **Date**.

- **Type Framework.** Le type correspondant dans le .NET Framework est la structure **System.DateTime**.

Exemple

Une variable ou constante de type de données **Date** contient à la fois la date et l'heure. L'exemple suivant illustre ce comportement :

```
Dim someDateAndTime As Date = #8/13/2002 12:14 PM#
```

Voir aussi

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
System.DateTime

Concepts

Utilisation efficace des types de données
Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Decimal, type de données (Visual Basic)

Contient des valeurs 128 bits (16 octets) signées représentant des entiers 96 bits (12 octets) mis à une puissance variable de 10. Le facteur d'échelle spécifie le nombre de chiffres situés à droite de la virgule décimale ; il est compris entre 0 et 28. Avec une échelle de 0 (aucune décimale), la plus grande valeur possible est +/-79 228 162 514 264 337 593 543 950 335 (+/-7,9228162514264337593543950335E+28). Avec 28 décimales, la plus grande valeur est +/-7,9228162514264337593543950335, et la plus petite valeur différente de zéro est +/-0,0000000000000000000000000001 (+/-1E-28).

Notes

Le type de données **Decimal** fournit le plus grand nombre de chiffres significatifs pour un nombre. Il prend en charge jusqu'à 29 chiffres significatifs et peut représenter des valeurs supérieures à *7.9228 x 10^28*. Ce type de données est particulièrement adapté aux calculs (par exemple, financiers) qui exigent un grand nombre de chiffres, mais qui ne peuvent pas tolérer les erreurs d'arrondi.

La valeur par défaut de **Decimal** est 0.

Conseils de programmation

- **Précision. Decimal** n'est pas un type de données à virgule flottante. La structure **Decimal** contient une valeur entière binaire, avec un bit de signe et un facteur d'échelle d'entier qui spécifie quelle partie de la valeur est une fraction décimale. C'est pourquoi les nombres **Decimal** ont une représentation plus précise dans la mémoire que les types à virgule flottante (**Single** et **Double**).
- **Performances.** Le type de données **Decimal** est le plus lent de tous les types numériques. Vous devez peser l'importance de la précision par rapport aux performances avant de choisir un type de données.
- **Extension.** Le type de données **Decimal** s'étend à **Single** ou **Double**. Cela signifie que vous pouvez convertir des données de type **Decimal** en données de l'un ou l'autre de ces types sans rencontrer d'erreur System.OverflowException.
- **Zéros de fin.** Visual Basic ne stocke pas de zéros de fin dans un littéral **Decimal**. Toutefois, une variable **Decimal** conserve tous les zéros de fin obtenus par le calcul. L'exemple suivant illustre ce comportement :

  ```
  Dim d1, d2, d3, d4 As Decimal
  d1 = 2.375D
  d2 = 1.625D
  d3 = d1 + d2
  d4 = 4.000D
  MsgBox("d1 = " & CStr(d1) & ", d2 = " & CStr(d2) _
      & ", d3 = " & CStr(d3) & ", d4 = " & CStr(d4))
  ```

  La sortie de **MsgBox** dans l'exemple précédent est la suivante :

  d1 = 2.375, d2 = 1.625, d3 = 4.000, d4 = 4

- **Caractères de type.** L'ajout du caractère de type de littéral **D** à un littéral force ce dernier en un type de données **Decimal**. L'ajout du caractère de type d'identificateur **@** à un identificateur force ce dernier en un type **Decimal**.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Decimal.

Plage

Il est parfois nécessaire d'utiliser le caractère de type **D** pour assigner une grande valeur à une variable ou à une constante **Decimal**. L'exemple suivant illustre ce comportement :

```
Dim bigDec1 As Decimal = 9223372036854775807   ' No overflow.
Dim bigDec2 As Decimal = 9223372036854775808   ' Overflow.
Dim bigDec3 As Decimal = 9223372036854775808D  ' No overflow.
```

À moins qu'il ne soit suivi d'un caractère de type de littéral, un littéral est interprété par le compilateur comme étant de type **Long**. La déclaration de `bigDec1` ne produit pas de dépassement de capacité, car sa valeur est comprise dans la plage de **Long**.

Toutefois, la valeur de `bigDec2` est trop grande pour **Long**, de sorte que le compilateur produit une erreur. Le caractère de type de littéral **D** résout le problème pour `bigDec3` en forçant le compilateur à interpréter le littéral en tant que **Decimal**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Single, type de données (Visual Basic)
Double, type de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
System.Decimal
System.Decimal.#ctor(System.Int32[])

Concepts

Utilisation efficace des types de données

Double, type de données (Visual Basic)

Contient des nombres à virgule flottante en double précision IEEE 64 bits (8 octets) signés dont la valeur est comprise entre -1,79769313486231570E+308 et -4,94065645841246544E-324 pour les valeurs négatives et entre 4,94065645841246544E-324 et 1,79769313486231570E+308 pour les valeurs positives. Les nombres en double précision stockent une approximation d'un nombre réel.

Notes

Le type de données **Double** fournit les amplitudes les plus grandes et les plus petites possibles pour un nombre.

La valeur par défaut de **Double** est 0.

Conseils de programmation

- **Précision.** Lorsque vous utilisez des nombres à virgule flottante, pensez qu'ils n'ont pas toujours de représentation précise dans la mémoire. Certaines opérations pourraient avoir des résultats inattendus, comme la comparaison de valeur et l'opérateur **Mod**. Pour plus d'informations, consultez Dépannage des types de données.
- **Zéros de fin.** Les types de données à virgule flottante n'ont aucune représentation interne du 0 de fin. Par exemple, ils ne font pas la différence entre 4,2000 et 4,2. Par conséquent, le 0 de fin n'apparaît pas lorsque vous affichez ou imprimez des valeurs à virgule flottante.
- **Caractères de type.** L'ajout du caractère de type de littéral **R** à un littéral force ce dernier en un type de données **Double**. L'ajout du caractère de type d'identificateur **#** à un identificateur force ce dernier en un type **Double**.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Double.

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)
Decimal, type de données (Visual Basic)
Single, type de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
System.Double

Concepts

Utilisation efficace des types de données

Integer, type de données (Visual Basic)

Contient des entiers 32 bits (4 octets) signés dont la valeur est comprise entre -2 147 483 648 et 2 147 483 647.

Notes

Le type de données **Integer** offre des performances optimales sur un processeur de 32 bits. Les autres types intégraux sont plus lents à charger et à stocker à partir de et dans la mémoire.

La valeur par défaut de **Integer** est 0.

Conseils de programmation

- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, par exemple des objets Automation ou COM, n'oubliez pas que **Integer** possède une largeur des données différente (16 bits) dans d'autres environnements. Si vous passez un argument de 16 bits à un tel composant, déclarez-le comme type de données **Short** et non comme **Integer** dans votre nouveau code Visual Basic.
- **Extension.** Le type de données **Integer** s'étend à **Long**, **Decimal**, **Single** ou **Double**. Ceci signifie que vous pouvez convertir **Integer** en ces types sans rencontrer d'erreur System.OverflowException.
- **Caractères de type.** L'ajout du caractère de type de littéral **I** à un littéral force ce dernier en un type de données **Integer**. L'ajout du caractère de type d'identificateur **%** à un identificateur force ce dernier en un type **Integer**.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Int32.

Plage

Si vous essayez de définir une variable d'un type intégral avec un nombre situé hors de la plage de ce type, une erreur se produit. Si vous essayez de lui affecter une fraction, le nombre est arrondi. L'exemple suivant fournit une illustration.

```
' The valid range of an Integer variable is -2147483648 through +2147483647.
Dim k As Integer
' The following statement causes an error because the value is too large.
k = 2147483648
' The following statement sets k to 6.
k = CInt(5.9)
```

Voir aussi

Référence

Liste des types de données (Visual Basic)
Long, type de données (Visual Basic)
Short, type de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
System.Int32

Concepts

Type de données Integer pour les utilisateurs de Visual Basic 6.0
Utilisation efficace des types de données

Long, type de données (Visual Basic)

Contient des entiers 64 bits (8 octets) signés dont la valeur est comprise entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807 (9 2...E+18).

Notes

Utilisez le type de données **Long** pour contenir des nombres entiers qui sont trop grands pour le type de données **Integer**.

La valeur par défaut de **Long** est 0.

Conseils de programmation

- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, par exemple des objets Automation ou COM, n'oubliez pas que **Long** possède une largeur des données différente (32 bits) dans d'autres environnements. Si vous passez un argument de 32 bits à un tel composant, déclarez-le comme type de données **Integer** et non comme **Long** dans votre nouveau code Visual Basic.

  En outre, Automation ne prend pas en charge des entiers 64 bits sous Windows 95, Windows 98, Windows ME ou Windows 2000. Vous ne pouvez pas passer un argument **Long** Visual Basic à un composant Automation sur de telles plates-formes.

- **Extension.** Le type de données **Long** s'étend à **Decimal**, **Single** ou **Double**. Ceci signifie que vous pouvez convertir **Byte** en ces types sans rencontrer d'erreur System.OverflowException.

- **Caractères de type.** L'ajout du caractère de type de littéral **L** à un littéral force ce dernier en un type de données **Long**. L'ajout du caractère de type d'identificateur **&** à un identificateur force ce dernier en un type **Long**.

- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Int64.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Int64 Structure
Integer, type de données (Visual Basic)
Short, type de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions

Concepts

Type de données Integer pour les utilisateurs de Visual Basic 6.0
Utilisation efficace des types de données

Object, type de données

Contient des adresses 32 bits (4 octets) qui font référence à des objets. Vous pouvez assigner un type référence (chaîne, tableau, classe ou interface) à une variable **Object**. Une variable **Object** peut également faire référence à des données de tout type valeur (numérique, **Boolean**, **Char**, **Date**, structure ou énumération).

Notes

Le type de données **Object** peut pointer vers des données de tout type de données, y compris toute instance d'objet que votre application reconnaît. Utilisez **Object** lorsque vous ne savez pas au moment de la compilation quel est le type de données vers lequel la variable peut pointer.

La valeur par défaut de **Object** est **Nothing** (référence nulle).

Types de données

Vous pouvez assigner une variable, une constante ou une expression de n'importe quel type de données à une variable **Object**. Pour déterminer le type de données auquel une variable **Object** fait actuellement référence, vous pouvez utiliser la méthode GetTypeCode de la classe System.Type. L'exemple suivant illustre ce comportement :

```
Dim myObject As Object
' Suppose myObject has now had something assigned to it.
Dim datTyp As Integer
datTyp = Type.GetTypeCode(myObject.GetType())
```

Le type de données **Object** est un type référence. Toutefois, Visual Basic traite une variable **Object** comme un type valeur lorsqu'elle fait référence aux données d'un type valeur.

Stockage

Quel que soit le type de données auquel elle fait référence, une variable **Object** ne contient pas la valeur de donnée elle-même, mais plutôt un pointeur vers la valeur. Elle occupe toujours quatre octets dans la mémoire de l'ordinateur ; toutefois, ces quatre octets ne comprennent pas le stockage des données représentant la valeur de la variable. Étant donné que le code utilise le pointeur pour localiser les données, les variables **Object** contenant des types valeur sont légèrement plus lentes que celles qui sont explicitement typées.

Conseils de programmation

- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, tels que des objets Automation ou COM, gardez à l'esprit que les types pointeur dans d'autres environnements ne sont pas compatibles avec le type **Object** Visual Basic.
- **Performances.** Une variable que vous déclarez avec le type **Object** est assez flexible pour contenir une référence à n'importe quel objet. Toutefois, lorsque vous appelez une méthode ou une propriété sur une variable de ce type, vous subissez toujours la *liaison tardive* (au moment de l'exécution). Pour forcer la *liaison anticipée* (au moment de la compilation) et obtenir de meilleures performances, déclarez la variable avec un nom de classe spécifique ou effectuez un cast de celle-ci en type de données spécifique.

  Lorsque vous déclarez une variable objet, essayez d'utiliser un type de classe spécifique, par exemple OperatingSystem au lieu du type **Object** généralisé. Vous devez également utiliser la classe la plus spécifique parmi les classes disponibles, telle que TextBox au lieu de Control, de façon à pouvoir accéder à ses propriétés et méthodes. Vous pouvez généralement utiliser la liste **Classes** dans l'**Explorateur d'objets** pour rechercher le nom des classes disponibles.
- **Extension.** Tous les types de données et tous les types référence s'étendent au type de données **Object**. Cela signifie que vous pouvez convertir n'importe quel type en **Object** sans rencontrer d'erreur System.OverflowException.

  Toutefois, si vous effectuez une conversion entre des types valeur et **Object**, Visual Basic exécute des opérations appelées *conversion boxing* et *conversion unboxing* qui ralentissent l'exécution.
- **Caractères de type. Object** n'a aucun caractère de type de littéral ou caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la classe System.Object.

Exemple

L'exemple suivant illustre une variable **Object** pointant vers une instance d'objet.

```
Dim objDb As Object
Dim myCollection As New Collection()
' Suppose myCollection has now been populated.
objDb = myCollection.Item(1)
```

Voir aussi

Tâches

Comment : déterminer si deux objets sont liés

Comment : déterminer si deux objets sont identiques

Référence

Liste des types de données (Visual Basic)

Object Class

Fonctions de conversion de types de données

Liste des conversions

Concepts

Utilisation efficace des types de données

Object comme type de données universel

SByte, type de données (Visual Basic)

Contient des entiers 8 bits (1 octet) signés dont la valeur est comprise entre -128 et 127.

Notes

Utilisez le type de données **SByte** pour contenir des valeurs entières qui ne requièrent pas la largeur totale des données de **Integer** ou même la moitié de la largeur des données de **Short**. Dans certains cas, le Common Language Runtime peut être capable de comprimer les variables **SByte** et d'enregistrer la consommation de mémoire.

La valeur par défaut de **SByte** est 0.

Conseils de programmation

- **Conformité CLS.** Le type de données **SByte** ne faisant pas partie de Spécification CLS (Common Language Specification) (CLS), le code conforme CLS ne peut pas consommer un composant qui l'utilise.
- **Extension.** Le type de données **SByte** s'étend à **Short**, **Integer**, **Long**, **Decimal**, **Single** et **Double**. Ceci signifie que vous pouvez convertir **SByte** en ces types sans rencontrer d'erreur System.OverflowException.
- **Caractères de type. SByte** n'a aucun caractère de type de littéral ou caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.SByte.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
Short, type de données (Visual Basic)
Integer, type de données (Visual Basic)
Long, type de données (Visual Basic)
System.SByte

Concepts

Type de données Integer pour les utilisateurs de Visual Basic 6.0
Utilisation efficace des types de données

Short, type de données (Visual Basic)

Contient des entiers 16 bits (2 octets) signés dont la valeur est comprise entre -32,768 et 32,767.

Notes

Utilisez le type de données **Short** pour contenir des valeurs entières qui ne nécessitent pas la largeur complète des données de type **Integer**. Dans certains cas, le Common Language Runtime peut être capable de comprimer les variables **Short** ensemble et d'enregistrer la consommation de mémoire.

La valeur par défaut de **Short** est 0.

Conseils de programmation

- **Extension.** Le type de données **Short** s'étend à **Integer**, **Long**, **Decimal**, **Single** ou **Double**. Ceci signifie que vous pouvez convertir **Short** en ces types sans rencontrer d'erreur System.OverflowException.
- **Caractères de type.** L'ajout de caractères de type de littéral **S** à un littéral force ce dernier en type de données **Short**. **Short** ne possède pas de caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Int16.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
Integer, type de données (Visual Basic)
Long, type de données (Visual Basic)
System.Int16

Concepts

Type de données Integer pour les utilisateurs de Visual Basic 6.0
Utilisation efficace des types de données

Single, type de données (Visual Basic)

Contient des nombres à virgule flottante en simple précision IEEE 32 bits (4 octets) signés dont la valeur est comprise entre -3,4028235E+38 et -1,401298E-45 pour les valeurs négatives et entre 1,401298E-45 et 3,4028235E+38 pour les valeurs positives. Les nombres en simple précision stockent une approximation d'un nombre réel.

Notes

Utilisez le type de données **Single** pour contenir des valeurs à virgule flottante qui ne nécessitent pas la largeur complète des données de type **Double**. Dans certains cas, le Common Language Runtime peut être capable de comprimer les variables **Single** ensemble et d'enregistrer la consommation de mémoire.

La valeur par défaut de **Single** est 0.

Conseils de programmation

- **Précision.** Lorsque vous utilisez des nombres à virgule flottante, pensez qu'ils n'ont pas toujours de représentation précise dans la mémoire. Certaines opérations pourraient avoir des résultats inattendus, comme la comparaison de valeur et l'opérateur **Mod**. Pour plus d'informations, consultez Dépannage des types de données.
- **Extension.** Le type de données **Single** s'étend à **Double**. Cela signifie que vous pouvez convertir **Single** en **Double** sans produire d'erreur System.OverflowException.
- **Zéros de fin.** Les types de données à virgule flottante n'ont aucune représentation interne du 0 de fin. Par exemple, ils ne font pas la différence entre 4,2000 et 4,2. Par conséquent, le 0 de fin n'apparaît pas lorsque vous affichez ou imprimez des valeurs à virgule flottante.
- **Caractères de type.** L'ajout du caractère de type de littéral **F** à un littéral force ce dernier en un type de données **Single**. L'ajout du caractère de type d'identificateur **!** à un identificateur force ce dernier en un type **Single**.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.Single.

Voir aussi

Tâches

Dépannage des types de données

Référence

Liste des types de données (Visual Basic)
Decimal, type de données (Visual Basic)
Double, type de données (Visual Basic)
Fonctions de conversion de types de données
Liste des conversions
System.Single

Concepts

Utilisation efficace des types de données

String, type de données (Visual Basic)

Contient des séquences de points de code 16 bits (2 octets) non signés dont la valeur est comprise entre 0 et 65 535. Chaque *point de code*, ou code de caractère, représente un seul caractère Unicode. Une chaîne peut contenir jusqu'à environ 2 milliards (2 ^ 31) de caractères Unicode.

Notes

Utilisez le type de données **String** pour maintenir plusieurs caractères sans les charges mémoire de la gestion du tableau de **Char()**, un tableau d'éléments **Char**.

La valeur par défaut de **String** est **Nothing** (référence nulle). Notez que ce n'est pas la même valeur que la chaîne vide (valeur **""**).

Caractères Unicode

Les premiers 128 points de code (0–127) Unicode correspondent aux lettres et aux symboles du clavier américain standard. Ce sont les mêmes que ceux définis par le jeu de caractères ASCII. Les 128 points de code suivants (128–255) représentent les caractères spéciaux, tels que les lettres de l'alphabet latin, les accents, les symboles monétaires et les fractions. Unicode utilise les points de code restants (256-65 535) pour une large gamme de symboles, y compris les caractères textuels mondiaux, les signes diacritiques, ainsi que les symboles mathématiques et techniques.

Vous pouvez utiliser des méthodes telles que IsDigit et IsPunctuation sur un caractère dans une variable **String** pour déterminer sa classification Unicode.

Pré-requis de format

Vous devez joindre un littéral **String** entre des guillemets (**" "**). Si vous devez inclure un guillemet en tant que caractère dans la chaîne, utilisez deux guillemets contigus (**""**). L'exemple suivant illustre ce comportement :

```
Dim j As String = "Joe said ""Hello"" to me."
Dim h As String = "Hello"
' The following messages all display the same thing:
' "Joe said "Hello" to me."
MsgBox(j)
MsgBox("Joe said " & """ & h & """ & " to me.")
MsgBox("Joe said """ & h & """ to me.")
```

Notez que les guillemets contigus qui représentent un guillemet dans la chaîne sont indépendants des guillemets qui commencent et terminent le littéral **String**.

Manipulations de chaînes

Une fois que vous assignez une chaîne à une variable **String**, cette chaîne est *immuable*, ce qui signifie que vous ne pouvez pas modifier sa longueur ou son contenu. Lorsque vous modifiez une chaîne d'une manière ou d'une autre, Visual Basic crée une nouvelle chaîne et abandonne la précédente. La variable **String** pointe ensuite vers la nouvelle chaîne.

Vous pouvez manipuler le contenu d'une variable **String** à l'aide de diverses fonctions de chaîne. L'exemple suivant illustre le Left, fonction (Visual Basic).

```
Dim S As String = "Database"
' The following statement sets S to a new string containing "Data".
S = Microsoft.VisualBasic.Left(S, 4)
```

Une chaîne créée par un autre composant peut être remplie avec des espaces à gauche ou à droite. Si vous recevez une telle chaîne, vous pouvez utiliser le Fonctions Trim, LTrim et Rtrim pour supprimer ces espaces.

Pour plus d'informations sur les manipulations de chaînes, consultez Chaînes en Visual Basic.

Conseils de programmation

- **Nombres négatifs.** N'oubliez pas que les caractères stockés dans **String** ne sont pas signés et ne peuvent pas représenter de valeurs négatives. Dans tous les cas, vous ne devez pas utiliser **String** pour stocker des valeurs numériques.
- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, par exemple

des objets Automation ou COM, n'oubliez pas que les chaînes de caractères possèdent une largeur des données différente (8 bits) dans d'autres environnements. Si vous transmettez un argument string de caractères de 8 bits à un tel composant, déclarez-le comme **Byte()**, un tableau d'éléments **Byte**, au lieu de **String** dans votre nouveau code Visual Basic.

- **Caractères de type.** L'ajout de caractères de type identificateur **$** à un identificateur force ce dernier en un type de données **String**. **String** ne possède pas de caractères de type littéral. Toutefois, le compilateur assimile les littéraux compris entre guillemets (**" "**) en tant que **String**.
- **Type Framework.** Le type correspondant dans le .NET Framework est la classe System.String.

Voir aussi

Tâches

Comment : optimiser le stockage d'entiers positifs avec les types non signés

Comment : appeler une fonction Windows qui possède des types non signés

Référence

Liste des types de données (Visual Basic)

Char, type de données (Visual Basic)

Fonctions de conversion de types de données

Liste des conversions

System.String

Concepts

Utilisation efficace des types de données

UInteger, type de données

Contient des entiers 32 bits (4 octets) non signés dont la valeur est comprise entre 0 et 4 294 967 295.

Notes

Le type de données **UInteger** fournit la plus grande valeur non signée dans la largeur des données la plus efficace.

La valeur par défaut de **UInteger** est 0.

Conseils de programmation

Les types de données **UInteger** et **Integer** fournissent une performance optimale sur un processeur de 32 bits, car les plus petits types entiers (**UShort**, **Short**, **Byte** et **SByte**), bien qu'ils requièrent moins de bits, sont plus longs à charger, à stocker et à extraire.

Voir aussi

Tâches

Comment : optimiser le stockage d'entiers positifs avec les types non signés
Comment : appeler une fonction Windows qui possède des types non signés

Référence

Liste des types de données (Visual Basic)
UInt32 Structure
Fonctions de conversion de types de données
Liste des conversions

Concepts

Utilisation efficace des types de données

ULong, type de données (Visual Basic)

Contient des entiers 64 bits (8 octets) non signés dont la valeur est comprise entre 0 et 18 446 744 073 709 551 615 (plus que 1,84 fois 10 ^ 18).

Notes

Utilisez le type de données **ULong** pour stocker les données binaires trop grandes pour **UInteger** ou les plus grandes valeurs entières non signées possibles.

La valeur par défaut de **ULong** est 0.

Conseils de programmation

- **Nombres négatifs.** Dans la mesure où **ULong** est un type non signé, il ne peut représenter un nombre négatif. Si vous utilisez l'opérateur moins unaire (**-**) dans une expression qui correspond au type **ULong**, Visual Basic convertit d'abord l'expression en **Decimal**.
- **Conforme CLS.** Le type de données **ULong** ne faisant pas partie de Spécification CLS (Common Language Specification), le code conforme CLS ne peut pas consommer un composant qui l'utilise.
- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, par exemple des objets Automation ou COM, n'oubliez pas que les types tels que **ulong** peuvent avoir une largeur des données différente (32 bits) dans d'autres environnements. Si vous passez un argument de 32 bits à un tel composant, déclarez-le comme type de données **UInteger** et non comme **ULong** dans votre code managé Visual Basic .NET.

  En outre, Automation ne prend pas en charge des entiers 64 bits sous Windows 95, Windows 98, Windows ME ou Windows 2000. Vous ne pouvez pas passer un argument **ULong** Visual Basic à un composant Automation sur de telles plates-formes.
- **Extension.** Le type de données **ULong** s'étend à **Decimal**, **Single** et **Double**. Ceci signifie que vous pouvez convertir **ULong** en ces types sans rencontrer d'erreur System.OverflowException.
- **Caractères de type.** L'ajout de caractères de type littéral **UL** à un littéral force ce dernier en un type de données **ULong**. **ULong** ne possède pas de caractère de type d'identificateur.
- **Type Framework.** Le type correspondant dans le .NET Framework est la structure System.UInt64.

Voir aussi

Tâches

Comment : appeler une fonction Windows qui possède des types non signés

Référence

Liste des types de données (Visual Basic)
UInt64 Structure
Fonctions de conversion de types de données
Liste des conversions

Concepts

Utilisation efficace des types de données

Type de données défini par l'utilisateur

Stocke les données dans un format que vous définissez. L'instruction **Structure** définit le format.

Les versions antérieures de Visual Basic prennent en charge le type défini par l'utilisateur (TDU). La version actuelle développe le TDU en une *structure*. Une structure est une concaténation d'un ou plusieurs *membres* de divers types de données. Visual Basic traite une structure comme une unité unique, bien que vous puissiez également accéder à ses membres individuellement.

Notes

Définissez et utilisez un type de données de structure lorsque vous devez combiner divers types de données en une unité unique ou lorsque aucun des types de données élémentaires ne correspond à vos besoins.

La valeur par défaut d'un type de données de structure se compose de la combinaison des valeurs par défaut de chacun de ses membres.

Format de déclaration

Une déclaration de structure démarre par l'instruction Structure et se termine par l'instruction **End Structure**. L'instruction **Structure** fournit le nom de la structure, qui représente également l'identificateur du type de données défini par la structure. D'autres parties du code peuvent utiliser cet identificateur pour déclarer que les variables, paramètres et valeurs de retour des fonctions appartiennent à ce type de données de structure.

Les déclarations figurant entre les instructions **Structure** et **End Structure** définissent les membres de la structure.

Niveaux d'accès au membre

Vous devez déclarer chaque membre à l'aide d'une instruction Dim, instruction (Visual Basic) ou d'une instruction qui spécifie le niveau d'accès, comme Public (Visual Basic), Friend (Visual Basic)ou Private (Visual Basic). Si vous utilisez une instruction **Dim**, le niveau d'accès prend par défaut la valeur public.

Conseils de programmation

- **Consommation de mémoire.** Comme dans le cas de tous les types de données composites, vous ne pouvez pas calculer en toute sécurité la consommation totale de la mémoire d'une structure en additionnant les allocations de stockage nominal de ses membres. De plus, il est risqué de supposer que l'ordre de stockage dans la mémoire est identique à l'ordre de déclaration. Si vous devez contrôler la disposition de stockage d'une structure, vous pouvez appliquer l'attribut StructLayoutAttribute à l'instruction **Structure**.
- **Considérations sur l'interopérabilité.** Si vous utilisez des composants non écrits pour le .NET Framework, tels que des objets Automation ou COM, gardez à l'esprit que les types définis par l'utilisateur dans d'autres environnements ne sont pas compatibles avec les types de structure Visual Basic.
- **Extension.** Aucune conversion automatique n'est effectuée à partir ou vers un type de données de structure quelconque. Vous pouvez définir des opérateurs de conversion sur votre structure à l'aide de l'instruction Operator et pouvez déclarer chaque opérateur de conversion comme **Widening** ou **Narrowing**.
- **Caractères de type.** Les types de données de structure n'ont aucun caractère de type de littéral ou caractère de type d'identificateur.
- **Type Framework.** Il n'y a aucun type correspondant dans le .NET Framework. Toutes les structures héritent de la classe .NET Framework System.ValueType, mais aucune structure individuelle ne correspond à **System.ValueType**.

Exemple

Le paradigme suivant montre le plan de la déclaration d'une structure.

```
[Public | Protected | Friend | Protected Friend | Private] Structure structname
    {Dim | Public | Friend | Private} member1 As datatype1
    ' ...
    {Dim | Public | Friend | Private} memberN As datatypeN
End Structure
```

Voir aussi

Référence

Liste des types de données (Visual Basic)
ValueType Class
Fonctions de conversion de types de données
Liste des conversions
Structure, instruction
Widening
Narrowing
StructLayoutAttribute

Concepts

Utilisation efficace des types de données

Autres ressources

Structures constituées à partir de vos propres types de données

UShort, type de données (Visual Basic)

Contient des entiers 16 bits (2 octets) non signés dont la valeur est comprise entre 0 et 65 535.

Notes

Utilisez le type de données **UShort** pour contenir les données binaires trop grandes pour **Byte**.

La valeur par défaut de **UShort** est 0.

Conseils de programmation

- **Nombres négatifs.** Dans la mesure où **UShort** est un type non signé, il ne peut représenter un nombre négatif. Si vous utilisez l'opérateur moins unaire (**-**) dans une expression qui correspond au type **UShort**, Visual Basic convertit d'abord l'expression en **Integer**.
- **Conforme CLS.** Le type de données **UShort** ne faisant pas partie de Spécification CLS (Common Language Specification), le code conforme CLS ne peut pas consommer un composant qui l'utilise.
- **Extension.** Le type de données **UShort** s'étend à **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single**et **Double**. Ceci signifie que vous pouvez convertir **UShort** en ces types sans rencontrer d'erreur System.OverflowException.
- **Caractères de type.** L'ajout de caractères de type littéral **US** à un littéral force ce dernier en un type de données **UShort**. **UShort** ne possède pas de caractère de type d'identificateur.
- **Type France** Le type correspondant dans le .NET Framework est la structure System.UInt16.

Voir aussi

Tâches

Comment : appeler une fonction Windows qui possède des types non signés

Référence

Liste des types de données (Visual Basic)
UInt16 Structure
Fonctions de conversion de types de données
Liste des conversions

Concepts

Utilisation efficace des types de données

Directives (Visual Basic)

Les rubriques de cette section documentent les directives du compilateur de code source Visual Basic.

Dans cette section

#Const, directive

#ExternalSource, directive

#If...Then...#Else, directives

#Region, directive

Rubriques connexes

Référence Visual Basic

Visual Basic

#Const, directive

Définit les constantes de compilation conditionnelle de Visual Basic.

```
#Const constname = expression
```

Éléments

constname

Requis. **String**. Nom de la constante en cours de définition.

expression

Requis. Caractère littéral, autre constante de compilation conditionnelle, ou toute combinaison qui comprend tout ou partie des opérateurs arithmétiques ou logiques à l'exception de **Is**.

Notes

Les constantes de compilation conditionnelle sont toujours privées, c'est-à-dire qu'elles appartiennent au fichier dans lequel elles apparaissent. Vous ne pouvez pas créer de constantes de compilation publiques à l'aide de la directive **#Const** ; vous ne pouvez les créer que dans l'interface utilisateur ou à l'aide de l'option **/define** du compilateur.

Vous ne pouvez utiliser que des constantes de compilation conditionnelle et des littéraux dans *expression*. L'utilisation d'une constante standard définie avec **Const** génère une erreur. À l'inverse, vous ne pouvez utiliser des constantes définies avec le mot clé **#Const** que pour la compilation conditionnelle. Les constantes peuvent également être non définies. Dans ce cas, elles ont la valeur **Nothing**.

Exemple

Cet exemple utilise la directive **#Const**.

VB

```
#Const MyLocation = "USA"
#Const Version = "8.0.0012"
#Const CustomerNumber = 36
```

Voir aussi

Tâches

Comment : déclarer des constantes de compilation conditionnelle

Référence

/define (Visual Basic)

#If...Then...#Else, directives

Const, instruction (Visual Basic)

If...Then...Else, instruction (Visual Basic)

Concepts

Vue d'ensemble de la compilation conditionnelle

#ExternalSource, directive

Indique un mappage entre les lignes spécifiques du code source et le texte extérieur à la source.

```
#ExternalSource( StringLiteral , IntLiteral )
    [ LogicalLine+ ]
#End ExternalSource
```

Éléments

StringLiteral

Le chemin d'accès à la source externe.

IntLiteral

Le numéro de la première ligne de la source externe.

LogicalLine

La ligne de la source externe à laquelle l'erreur s'est produite.

#End ExternalSource

Met fin au bloc **#ExternalSource**.

Notes

Cette directive n'est utilisée que par le compilateur et le débogueur.

Un fichier source peut inclure des directives de source externe, qui indiquent un mappage entre les lignes spécifiques de code dans le fichier source et le texte externe à la source, tel qu'un fichier .aspx. Si des erreurs sont rencontrées dans le code source désigné lors de la compilation, elles sont identifiées comme émanant de la source externe.

Ces directives de source externe n'ont aucun effet sur la compilation et ne peuvent pas être imbriquées. Elles sont réservées à une utilisation interne par l'application.

Voir aussi

Autres ressources

Compilation conditionnelle (Visual Basic)

#If...Then...#Else, directives

Effectue une compilation conditionnelle des blocs de code Visual Basic sélectionnés.

```
#If expression Then
      statements
[ #ElseIf expression Then
   [ statements ]
...
#ElseIf expression Then
   [ statements ] ]
[ #Else
   [ statements ] ]
#End If
```

Éléments

expression

Requis pour les instructions **#If** et **#ElseIf**, sinon, facultatives. Toute expression, exclusivement constituée d'une ou plusieurs constantes de compilation conditionnelle, d'un ou plusieurs caractères littéraux et d'opérateurs, qui prend la valeur **True** ou **False**.

statements

Requis pour le bloc d'instruction **#If**, sinon, facultatifs. Lignes de programme ou directives de compilation Visual Basic compilées si l'expression associée prend la valeur **True**.

#End If

Met fin au bloc d'instruction **#If**.

Notes

En apparence, le comportement des directives **#If...Then...#Else** semble identique à celui des instructions **If...Then...Else**. Cependant, les directives **#If...Then...#Else** évaluent ce qui est compilé par le compilateur, alors que les instructions **If...Then...Else** évaluent les conditions au moment de l'exécution.

La compilation conditionnelle est généralement utilisée pour compiler le même programme sur différentes plates-formes. Elle est également utilisée pour éviter que le code de débogage n'apparaisse dans un fichier exécutable. Le code exclu lors d'une compilation conditionnelle est totalement absent du fichier exécutable final, et n'a donc aucune incidence sur la taille de ce dernier ou sur les performances.

Quel que soit le résultat des évaluations, toutes les expressions sont évaluées à l'aide de **Option Compare Binary**. L'instruction **Option Compare** n'affecte pas les expressions contenues dans les instructions **#If** et **#ElseIf**.

Remarque

Les directives **#If**, **#Else**, **#ElseIf** et **#End If** n'existent pas sous la forme d'une ligne unique. Aucun autre code ne peut figurer sur la même ligne que les directives.

Exemple

Cet exemple utilise la construction **#If...Then...#Else** pour déterminer la nécessité de compiler certaines instructions.

VB

```
#Const CustomerNumber = 36
#If CustomerNumber = 35 Then
        ' Insert code to be compiled for customer # 35.
#ElseIf CustomerNumber = 36 Then
        ' Insert code to be compiled for customer # 36.
#Else
        ' Insert code to be compiled for all other customers.
#End If
```

Voir aussi

Référence

#Const, directive

If...Then...Else, instruction (Visual Basic)

Constantes de compilation conditionnelle

Concepts

Vue d'ensemble de la compilation conditionnelle

#Region, directive

Réduit et masque des sections de code dans les fichiers Visual Basic.

```
#Region "identifier_string"
#End Region
```

Éléments
identifier_string

Requis. Chaîne qui sert de titre pour une zone réduite. Les zones sont réduites par défaut.

#End Region

Met fin au bloc **#Region**.

Notes

Utilisez la directive **#Region** pour spécifier un bloc de code à développer ou à réduire lors de l'utilisation de la fonctionnalité mode Plan de l'éditeur de code de Visual Studio. L'instruction **#Region** prend en charge la sémantique de bloc (telle que **#If**...**#End If**), ce qui signifie que le début et la fin doivent se trouver dans le même bloc de code.

Exemple

Cet exemple utilise la directive **#Region**.

VB

```
#Region "MathFunctions"
    ' Insert code for the Math functions here.
#End Region
```

Voir aussi
Tâches
Comment : structurer et masquer le code
Comment : réduire et masquer des sections de code
Référence
#If...Then...#Else, directives

Fonctions (Visual Basic)

Les rubriques de cette section contiennent des tables des fonctions membres runtime Visual Basic.

Dans cette section

Fonctions de conversion (Visual Basic)

Fonctions mathématiques (Visual Basic)

Fonctions de conversion de types de données

Fonctions A-C

Fonctions D-G

Fonctions H-L

Fonctions M-R

Fonctions S-Z

Rubriques connexes

Référence Visual Basic

Visual Basic

Fonctions de conversion (Visual Basic)

Asc, fonction

AscW, fonction

CBool, fonction

CByte, fonction

CChar, fonction

CDate, fonction

CDbl, fonction

CDec, fonction

Chr, fonction

ChrW, fonction

CInt, fonction

CLng, fonction

CObj, fonction

CSByte, fonction

CShort, fonction

CSng, fonction

CStr, fonction

CType, fonction

CUInt, fonction

CULng, fonction

CUShort, fonction

Format, fonction

Hex, fonction

Oct, fonction

Str, fonction

Val, fonction

Voir aussi

Référence

Fonctions de conversion de types de données

Fonctions mathématiques (Visual Basic)

Les fonctions mathématiques de Visual Basic 6 ont été remplacées par des méthodes équivalentes de la classe System.Math du .NET Framework.

Notes

Les méthodes mathématiques du .NET Framework sont identiques à leurs équivalents Visual Basic 6 d'un point de vue fonctionnel, bien que certaines aient des noms légèrement différents. Par exemple, l'équivalent .NET Framework de la fonction **Atn** Visual Basic 6 est Atan. Le tableau suivant dresse la liste des noms de fonctions mathématiques Visual Basic 6 et des méthodes .NET Framework équivalentes.

Fonction Visual Basic 6	Méthode .NET Framework	Description
Abs	Abs	Retourne la valeur absolue d'un nombre spécifié.
Atn	**Atan**	Retourne une valeur **Double** contenant l'angle dont la tangente est le nombre spécifié.
Cos	Cos	Retourne une valeur **Double** contenant le cosinus de l'angle spécifié.
Exp	Exp	Retourne une valeur **Double** contenant *e* (base des logarithmes népériens) élevé à la puissance spécifiée.
Log	Log	Retourne une valeur **Double** contenant le logarithme d'un nombre spécifié. Cette méthode est surchargée et peut retourner soit le logarithme népérien (de base *e*) d'un nombre spécifié, soit le logarithme d'un nombre spécifié dans une base spécifiée.
Round	Round	Retourne une valeur **Double** contenant le nombre le plus proche de la valeur spécifiée. Des fonctions d'arrondi supplémentaires sont disponibles en tant que méthodes de type intrinsèque, telles que Round.
Sgn	Sign	Retourne une valeur **Integer** indiquant le signe d'un nombre.
Sin	Sin	Retourne une valeur **Double** spécifiant le sinus d'un angle.
Sqr	Sqrt	Retourne une valeur **Double** spécifiant la racine carrée d'un nombre.
Tan	Tan	Retourne une valeur **Double** contenant la tangente d'un angle.

En outre, la classe mathématique .NET Framework fournit des constantes et autres méthodes statiques pour les fonctions trigonométriques, logarithmiques et les fonctions mathématiques courantes. Toutes ces méthodes peuvent être utilisées dans un programme Visual Basic.

Pour utiliser ces fonctions sans qualification, importez l'espace de noms **System.Math** dans votre projet en ajoutant le code suivant en haut du code source :

```
'Imports System.Math
```

Exemple

Cet exemple utilise la méthode **Abs** de la classe **Math** pour calculer la valeur absolue d'un nombre.

```
' Returns 50.3.
Dim MyNumber1 As Double = Math.Abs(50.3)
' Returns 50.3.
Dim MyNumber2 As Double = Math.Abs(-50.3)
```

Cet exemple utilise la méthode **Atan** de la classe **Math** pour calculer la valeur de pi.

```
Public Function GetPi() As Double
    ' Calculate the value of pi.
    Return 4.0 * Math.Atan(1.0)
End Function
```

Cet exemple utilise la méthode **Cos** de la classe **Math** pour retourner le cosinus d'un angle.

```
Public Function Sec(ByVal angle As Double) As Double
    ' Calculate the secant of angle, in radians.
    Return 1.0 / Math.Cos(angle)
End Function
```

Cet exemple utilise la méthode **Exp** de la classe **Math** pour retourner *e* élevé à une puissance.

```
Public Function Sinh(ByVal angle As Double) As Double
    ' Calculate hyperbolic sine of an angle, in radians.
    Return (Math.Exp(angle) - Math.Exp(-angle)) / 2.0
End Function
```

Cet exemple utilise la méthode **Log** de la classe **Math** pour retourner le logarithme népérien d'un nombre.

```
Public Function Asinh(ByVal value As Double) As Double
    ' Calculate inverse hyperbolic sine, in radians.
    Return Math.Log(value + Math.Sqrt(value * value + 1.0))
End Function
```

Cet exemple utilise la méthode **Round** de la classe **Math** pour arrondir un nombre au nombre entier le plus proche.

```
' Returns 3.
Dim MyVar2 As Double = Math.Round(2.8)
```

Cet exemple utilise la méthode **Sign** de la classe **Math** pour déterminer le signe d'un nombre.

```
' Returns 1.
Dim MySign1 As Integer = Math.Sign(12)
' Returns -1.
Dim MySign2 As Integer = Math.Sign(-2.4)
' Returns 0.
Dim MySign3 As Integer = Math.Sign(0)
```

Cet exemple utilise la méthode **Sin** de la classe **Math** pour retourner le sinus d'un angle.

```
Public Function Csc(ByVal angle As Double) As Double
    ' Calculate cosecant of an angle, in radians.
    Return 1.0 / Math.Sin(angle)
End Function
```

Cet exemple utilise la méthode **Sqrt** de la classe **Math** pour calculer la racine carrée d'un nombre.

```
' Returns 2.
Dim MySqr1 As Double = Math.Sqrt(4)
' Returns 4.79583152331272.
Dim MySqr2 As Double = Math.Sqrt(23)
' Returns 0.
Dim MySqr3 As Double = Math.Sqrt(0)
' Returns NaN (not a number).
Dim MySqr4 As Double = Math.Sqrt(-4)
```

Cet exemple utilise la méthode **Tan** de la classe **Math** pour retourner la tangente d'un angle.

```
Public Function Ctan(ByVal angle As Double) As Double
    ' Calculate cotangent of an angle, in radians.
    Return 1.0 / Math.Tan(angle)
End Function
```

Configuration requise

Classe : Math

Espace de noms : System

Assembly : mscorlib (dans mscorlib.dll)

Voir aussi

Référence

Rnd, fonction (Visual Basic)
Randomize, fonction (Visual Basic)
Fonctions mathématiques dérivées (Visual Basic)
NaN

Fonctions de conversion de types de données

Ces fonctions sont compilées "inline", c'est-à-dire que le code de conversion fait partie du code qui évalue l'expression. Parfois, aucune procédure n'est appelée pour effectuer la conversion, ce qui permet d'améliorer les performances. Chaque fonction convertit une expression en un type de données spécifique.

```
CBool(expression)
CByte(expression)
CChar(expression)
CDate(expression)
CDbl(expression)
CDec(expression)
CInt(expression)
CLng(expression)
CObj(expression)
CSByte(expression)
CShort(expression)
CSng(expression)
CStr(expression)
CUInt(expression)
CULng(expression)
CUShort(expression)
```

Élément
expression

Obligatoire. Toute expression du type de données source.

Type de données de la valeur de retour

Le nom de la fonction détermine le type de données de la valeur qu'elle retourne, comme l'illustre le tableau suivant.

Nom de la fonction	**Type de données retourné**	**Plage de l'argument *expression***
CBool	Boolean, type de données (Visual Basic)	Toute expression numérique, **Char** ou **String** valide.
CByte	Byte, type de données (Visual Basic)	0 à 255 (non signé) ; les parties fractionnaires sont arrondies.[1]
CChar	Char, type de données (Visual Basic)	Toute expression **Char** ou **String** valide ; seul le premier caractère d'un **String** est converti ; la valeur peut être comprise entre 0 et 65 535 (non signé).
CDate	Type de données date (Visual Basic)	Toute représentation valide de la date et de l'heure.
CDbl	Double, type de données (Visual Basic)	-1,79769313486231570E+308 à -4,94065645841246544E-324 pour les valeurs négatives ; 4,94065645841246544E-324 à 1,79769313486231570E+308 pour les valeurs positives.
CDec	Decimal, type de données (Visual Basic)	+/-79 228 162 514 264 337 593 543 950 335 pour les nombres sans décimales. La plage de valeurs des nombres à 28 décimales est +/-7,9228162514264337593543950335. Le plus petit nombre différent de zéro est 0,0000000000000000000000000001 (+/-1E-28).
CInt	Integer, type de données (Visual Basic)	-2 147 483 648 à 2 147 483 647 ; les parties fractionnaires sont arrondies.[1]
CLng	Long, type de données (Visual Basic)	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 ; les parties fractionnaires sont arrondies.[1]

CObj	Object, type de données	Toute expression valide.
CSByte	SByte, type de données (Visual Basic)	-128 à 127 ; les parties fractionnaires sont arrondies.[1]
CShort	Short, type de données (Visual Basic)	-32 768 à 32 767 ; les parties fractionnaires sont arrondies.[1]
CSng	Single, type de données (Visual Basic)	-3,402823E+38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E+38 pour les valeurs positives.
CStr	String, type de données (Visual Basic)	Les valeurs retournées pour **CStr** dépendent de l'argument *expression*. Consultez Valeurs de retour pour la fonction CStr.
CUInt	UInteger, type de données	0 à 4 294 967 295 (non signé) ; les parties fractionnaires sont arrondies.[1]
CULng	ULong, type de données (Visual Basic)	0 à 18 446 744 073 709 551 615 (non signé) ; les parties fractionnaires sont arrondies.[1]
CUShort	UShort, type de données (Visual Basic)	0 à 65 535 (non signé) ; les parties fractionnaires sont arrondies.[1]

[1] Les parties fractionnaires peuvent faire l'objet d'un type spécial d'arrondissement nommé *arrondi bancaire*. Consultez « Notes » pour obtenir plus d'informations.

Notes

En règle générale, vous devez utiliser les fonctions de conversion de Visual Basic de préférence aux méthodes .NET Framework, telles que `ToString()`, sur la classe Convert, sur une structure de type ou sur une classe. Les fonctions de Visual Basic sont conçues pour une interaction optimale avec le code Visual Basic ; elles permettent également de raccourcir votre code source et d'en faciliter la lecture. En outre, les méthodes de conversion .NET Framework ne produisent pas toujours les mêmes résultats que les fonctions Visual Basic, par exemple, lors de la conversion de **Boolean** en **Integer**. Pour plus d'informations, consultez Dépannage des types de données.

Comportement

- **Contrainte.** En général, vous pouvez utiliser des fonctions de conversion des types de données pour forcer les résultats d'une opération à correspondre à un type de données particulier au lieu du type de données par défaut. Par exemple, utilisez **CDec** pour fonctionner en arithmétique décimale et non en arithmétique en simple précision, en double précision ou en arithmétique sur des nombres entiers.
- **Échec des conversions.** Si le *expression* passé à la fonction se trouve en dehors de la plage du type de données dans lequel il doit être converti, un OverflowException se produit.
- **Parties fractionnaires.** Lorsque vous convertissez une valeur non entière dans un type entier, les fonctions de conversion d'entiers (**CByte**, **CInt**, **CLng**, **CSByte**, **CShort**, **CUInt**, **CULng** et **CUShort**) suppriment la partie fractionnaire et arrondissent la valeur au nombre entier le plus proche.

  Si la partie fractionnaire est exactement égale à 0,5, les fonctions de conversion d'entier l'arrondissent toujours au nombre entier pair le plus proche. Par exemple, 0,5 est arrondi à 0 ; 1,5 et 2,5 sont tous les deux arrondis à 2. C'est ce que l'on appelle parfois l'*arrondi bancaire* et son but est de compenser un écart qui pourrait s'accumuler lors de l'ajout de tels nombres.

  CInt et **CLng** diffèrent des fonctions Int, Fix (Visual Basic), qui tronquent, au lieu d'arrondir, la partie fractionnaire d'un nombre. De plus, **Fix** et **Int** retournent toujours une valeur du même type de données que celui que vous avez passé.
- **Conversions date/heure.** Utilisez la fonction IsDate (Visual Basic) pour déterminer si une valeur peut être convertie en une date et une heure. **CDate** reconnaît les littéraux date et heure, mais pas les valeurs numériques. Pour convertir une valeur **Date** Visual Basic 6.0 en une valeur **Date** Visual Basic 2005, vous pouvez utiliser la méthode System.DateTime.FromOADate(System.Double).
- **Valeurs date/heure neutres.** Le type de données Date (Visual Basic) contient toujours des informations sur la date et l'heure. Dans le cadre de la conversion de type, Visual Basic considère la valeur 1/1/0001 (le 1er janvier de l'année 1)

comme une *valeur neutre* pour la date et 00:00:00 (minuit) comme une valeur neutre pour l'heure. Si vous convertissez une valeur **Date** en une chaîne, **CStr** n'inclut pas de valeurs neutres dans la chaîne obtenue. Par exemple, si vous convertissez `#January 1, 0001 9:30:00#` en une chaîne, la chaîne obtenue est "9:30:00 AM" ; les informations relatives à la date sont supprimées. Toutefois, les informations de date sont toujours présentes dans la valeur **Date** d'origine et peuvent être récupérées à l'aide de fonctions, telles que la fonction DatePart (Visual Basic).

- **Respect de la culture.** Les fonctions de conversion de type impliquant des chaînes effectuent des conversions en fonction des paramètres de culture actuels de l'application. Par exemple, **CDate** reconnaît les formats de date définis dans les paramètres régionaux de votre système. Vous devez fournir le jour, le mois et l'année selon l'ordre correct pour vos paramètres régionaux ; sinon, la date pourrait être incorrectement interprétée. Les formats de date complets ne sont pas reconnus s'ils contiennent une chaîne précisant un jour de la semaine (par exemple, "mercredi").

  Si vous devez effectuer une conversion depuis ou vers une représentation sous forme de chaîne d'une valeur dans un format différent de celui spécifié dans vos paramètres régionaux, vous ne pouvez pas utiliser les fonctions de conversion de type Visual Basic. Pour cela, utilisez les méthodes `ToString(IFormatProvider)` et `Parse(String, IFormatProvider)` du type de cette valeur. Par exemple, utilisez System.Double.Parse(System.String,System.IFormatProvider) lors de la conversion d'une chaîne en un **Double** et utilisez System.Double.ToString(System.IFormatProvider) lors de la conversion d'une valeur de type **Double** en une chaîne.

CType, fonction

La fonction CType accepte un deuxième argument, *typename*, et force *expression* à correspondre à *typename*, où *typename* peut être n'importe quel type de données, structure, classe ou interface vers lequel une conversion valide existe.

Pour obtenir une comparaison entre **CType** et les autres mots clés de conversion de type, consultez DirectCast et TryCast.

CBool, exemple

L'exemple suivant utilise la fonction **CBool** pour convertir des expressions en valeurs **Boolean**. Si une expression prend une valeur différente de zéro, **CBool** retourne **True** ; sinon, elle retourne **False**.

VB

```
Dim a, b, c As Integer
Dim check As Boolean
a = 5
b = 5
' The following line of code sets check to True.
check = CBool(a = b)
c = 0
' The following line of code sets check to False.
check = CBool(c)
```

CByte, exemple

L'exemple suivant utilise la fonction **CByte** pour convertir une expression en un **Byte**.

VB

```
Dim aDouble As Double
Dim aByte As Byte
aDouble = 125.5678
' The following line of code sets aByte to 126.
aByte = CByte(aDouble)
```

CChar, exemple

L'exemple suivant utilise la fonction **CChar** pour convertir le premier caractère d'une expression **String** en un type **Char**.

VB

```
Dim aString As String
Dim aChar As Char
' CChar converts only the first character of the string.
aString = "BCD"
' The following line of code sets aChar to "B".
```

```
aChar = CChar(aString)
```

L'argument d'entrée de **CChar** doit être du type de données **Char** ou **String**. Vous ne pouvez pas utiliser **CChar** pour convertir un nombre en un caractère, car **CChar** ne peut pas accepter un type de données numérique. L'exemple suivant obtient un nombre représentant un point de code (code de caractère) et le convertit dans le caractère correspondant. Il utilise la fonction InputBox (Visual Basic) pour obtenir la chaîne de chiffres, **CInt** pour convertir la chaîne en type **Integer** et **ChrW** pour convertir le nombre en type **Char**.

VB

```
Dim someDigits As String
Dim codePoint As Integer
Dim thisChar As Char
someDigits = InputBox("Enter code point of character:")
codePoint = CInt(someDigits)
' The following line of code sets thisChar to the Char value of codePoint.
thisChar = ChrW(codePoint)
```

CDate, exemple

L'exemple suivant utilise la fonction **CDate** pour convertir des chaînes en valeurs **Date**. En général, les dates et les heures codées de manière irréversible comme des chaînes (comme l'illustre cet exemple) ne sont pas recommandées. Utilisez plutôt des littéraux de date et d'heure, tels que #Feb 12, 1969# et #4:45:23 PM#.

VB

```
Dim aDateString, aTimeString As String
Dim aDate, aTime As Date
aDateString = "February 12, 1969"
aTimeString = "4:35:47 PM"
' The following line of code sets aDate to a Date value.
aDate = CDate(aDateString)
' The following line of code sets aTime to Date value.
aTime = CDate(aTimeString)
```

CDbl, exemple

VB

```
Dim aDec As Decimal
Dim aDbl As Double
' The following line of code uses the literal type character D to make aDec a Decimal.
aDec = 234.456784D
' The following line of code sets aDbl to 1.9225456288E+1.
aDbl = CDbl(aDec * 8.2D * 0.01D)
```

CDec, exemple

L'exemple suivant utilise la fonction **CDec** pour convertir une valeur numérique en **Decimal**.

VB

```
Dim aDouble As Double
Dim aDecimal As Decimal
aDouble = 10000000.0587
' The following line of code sets aDecimal to 10000000.0587.
aDecimal = CDec(aDouble)
```

CInt, exemple

L'exemple suivant utilise la fonction **CInt** pour convertir une valeur en **Integer**.

VB

```
Dim aDbl As Double
Dim anInt As Integer
aDbl = 2345.5678
' The following line of code sets anInt to 2346.
anInt = CInt(aDbl)
```

CLng, exemple

L'exemple suivant utilise la fonction **CLng** pour convertir des valeurs en **Long**.

VB

```
Dim aDbl1, aDbl2 As Double
Dim aLng1, aLng2 As Long
aDbl1 = 25427.45
aDbl2 = 25427.55
' The following line of code sets aLng1 to 25427.
aLng1 = CLng(aDbl1)
' The following line of code sets aLng2 to 25428.
aLng2 = CLng(aDbl2)
```

CObj, exemple

L'exemple suivant utilise la fonction **CObj** pour convertir une valeur numérique en **Object**. La variable **Object** contient uniquement un pointeur de quatre octets, qui pointe vers la valeur **Double** qui lui est assignée.

VB

```
Dim aDouble As Double
Dim anObject As Object
aDouble = 2.7182818284
' The following line of code sets anObject to a pointer to aDouble.
anObject = CObj(aDouble)
```

Exemple CSByte

L'exemple suivant utilise la fonction **CSByte** pour convertir une valeur numérique en **SByte**.

VB

```
Dim aDouble As Double
Dim anSByte As SByte
aDouble = 39.501
' The following line of code sets anSByte to 40.
anSByte = CSByte(aDouble)
```

CShort, exemple

L'exemple suivant utilise la fonction **CShort** pour convertir une valeur numérique en **Short**.

VB

```
Dim aByte As Byte
Dim aShort As Short
aByte = 100
' The following line of code sets aShort to 100.
aShort = CShort(aByte)
```

CSng, exemple

L'exemple suivant utilise la fonction **CSng** pour convertir des valeurs en **Single**.

VB

```
Dim aDouble1, aDouble2 As Double
Dim aSingle1, aSingle2 As Single
aDouble1 = 75.3421105
aDouble2 = 75.3421567
' The following line of code sets aSingle1 to 75.34211.
aSingle1 = CSng(aDouble1)
' The following line of code sets aSingle2 to 75.34216.
aSingle2 = CSng(aDouble2)
```

CStr, exemple

L'exemple suivant utilise la fonction **CStr** pour convertir une valeur numérique en **String**.

VB

```
Dim aDouble As Double
Dim aString As String
aDouble = 437.324
' The following line of code sets aString to "437.324".
aString = CStr(aDouble)
```

L'exemple suivant utilise la fonction **CStr** pour convertir des valeurs **Date** en valeurs **String**.

VB

```
Dim aDate As Date
Dim aString As String
' The following line of code generates a COMPILER ERROR because of invalid format.
' aDate = #February 12, 1969 00:00:00#
' Date literals must be in the format #m/d/yyyy# or they are invalid.
' The following line of code sets the time component of aDate to midnight.
aDate = #2/12/1969#
' The following conversion suppresses the neutral time value of 00:00:00.
' The following line of code sets aString to "2/12/1969".
aString = CStr(aDate)
' The following line of code sets the time component of aDate to one second past midnight.
aDate = #2/12/1969 12:00:01 AM#
' The time component becomes part of the converted value.
' The following line of code sets aString to "2/12/1969 12:00:01 AM".
aString = CStr(aDate)
```

CStr retourne toujours une valeur **Date** au format de date courte des paramètres régionaux en cours, par exemple, "15/06/2003 16:35:47". Toutefois, **CStr** supprime les *valeurs neutres* de 1/1/0001 pour la date et 00:00:00 pour l'heure.

Pour plus d'informations sur les valeurs retournées par **CStr**, consultez Valeurs de retour pour la fonction CStr.

Exemple CUInt

L'exemple suivant utilise la fonction **CUInt** pour convertir une valeur numérique en **UInteger**.

VB

```
Dim aDouble As Double
Dim aUInteger As UInteger
aDouble = 39.501
' The following line of code sets aUInteger to 40.
aUInteger = CUInt(aDouble)
```

CULng, exemple

L'exemple suivant utilise la fonction **CULng** pour convertir une valeur numérique en **ULong**.

VB

```
Dim aDouble As Double
```

```
Dim aULong As ULong
aDouble = 39.501
' The following line of code sets aULong to 40.
aULong = CULng(aDouble)
```

CUShort, exemple

L'exemple suivant utilise la fonction **CUShort** pour convertir une valeur numérique en **UShort**.

VB

```
Dim aDouble As Double
Dim aUShort As UShort
aDouble = 39.501
' The following line of code sets aUShort to 40.
aUShort = CUShort(aDouble)
```

Voir aussi

Référence

Fonctions de conversion (Visual Basic)
Fonctions Asc, AscW
Fonctions Chr, ChrW
Fonctions Int, Fix (Visual Basic)
Format, fonction
Hex, fonction (Visual Basic)
Oct, fonction
Str, fonction
Val, fonction

Autres ressources

Conversions de type en Visual Basic

Valeurs de retour pour la fonction CStr

Le tableau suivant illustre les valeurs de retour de **CStr** pour différents types de données de *expression*.

Si le type *expression* a la valeur	**CStr retourne**
Boolean, type de données (Visual Basic)	Une chaîne contenant les valeurs "True" ou "False".
Type de données date (Visual Basic)	Une chaîne contenant une valeur **Date** (date et heure) dans le format de date courte e n vigueur sur votre système.
Types de données numériques	Une chaîne représentant le nombre.

CStr et Date

Le type **Date** contient toujours des informations sur la date et l'heure. Dans le cadre de la conversion de type, Visual Basic considère la valeur 1/1/0001 (le 1er janvier de l'année 1) comme une *valeur neutre* pour la date et 00:00:00 (minuit) comme une valeur neutre pour l'heure. **CStr** n'inclut pas de valeurs neutres dans la chaîne résultante. Par exemple, si vous convertissez `#January 1, 0001 9:30:00#` en une chaîne, la chaîne obtenue est "9:30:00 AM" ; les informations relatives à la date sont supprimées. Toutefois, les informations de date sont toujours présentes dans la valeur **Date** d'origine et peuvent être récupérées à l'aide de fonctions, telles que DatePart, fonction (Visual Basic).

Remarque
La fonction **CStr** effectue sa conversion en fonction des paramètres de culture actuels de l'application. Pour obtenir la repréS entation sous forme de chaîne d'un nombre dans une culture particulière, utilisez la méthode `ToString(IFormatProvider)` d u nombre. Par exemple, utilisez System.Double.ToString(System.IFormatProvider) lors de la conversion d'une valeur de type **Double** en **String**.

Voir aussi

Référence

Fonctions de conversion de types de données
Boolean, type de données (Visual Basic)
Type de données date (Visual Basic)
DatePart, fonction (Visual Basic)

Fonctions A-C

Le tableau suivant contient une liste des fonctions membres runtime Visual Basic.

AppActivate	Asc	AscW	Beep
CallByName	CBool	CByte	CChar
CDate	CDbl	CDec	ChDir
ChDrive	Choose	Chr	ChrW
CInt	CLng	CObj	Command
CreateObject	CShort	CSng	CStr
CType	CurDir		

Les fonctions mathématiques **Abs**, **Atn** et **Cos**, qui étaient disponibles en tant que fonctions membres runtime dans les versions 6.0 et antérieures de Visual Basic, ont été remplacées par des méthodes (Abs, Atan, Cos) de la classe Math de la bibliothèque de classes .NET Framework. Pour plus d'informations, consultez Fonctions mathématiques (Visual Basic).

Notes du développeur sur Smart Device

Les fonctions suivantes ne sont pas prises en charge dans les applications Smart Device : **AppActivate**, **Beep**, **CallByName**, **ChDir**, **ChDrive**, **CreateObject** et **CurDir**.

Voir aussi

Référence

Fonctions D-G
Fonctions H-L
Fonctions M-R
Fonctions S-Z

Autres ressources

Référence Visual Basic

AppActivate, fonction

Active une application qui est déjà en cours d'exécution.

```
Public Overloads Sub AppActivate( _
      ByVal { Title As String | ProcessId As Integer } _
)
```

Paramètres

Title

Expression **String** qui spécifie le titre affiché dans la barre de titre de l'application que vous souhaite activer. Vous pouvez utiliser le titre assigné à l'application lors de son lancement.

ProcessId

Integer qui spécifie le numéro d'identification de processus Win32 assigné à ce processus. Vous pouvez utiliser l'ID retourné par la Shell, fonction, à condition que sa valeur soit différente de zéro.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Title* ou *ProcessId* est introuvable.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Vous pouvez utiliser **AppActivate** pour donner le focus à la fenêtre active d'une application. Il se peut que vous ne possédiez pas de handle ou de référence à la fenêtre active, ou que vous ne sachiez pas quelle fenêtre est active à un moment donné. Dans ce cas, vous ne pouvez pas utiliser la méthode Focus.

La fonction **AppActivate** rend active l'application ou la fenêtre nommée, mais ne l'agrandit ni ne la réduit. Le focus quitte la fenêtre active de l'application lorsque l'utilisateur exécute une action visant à modifier le focus ou à fermer la fenêtre. Vous pouvez utiliser la fonction **Shell** pour démarrer une application et définir le style de la fenêtre.

Si vous employez le paramètre *Title*, **AppActivate** utilise une comparaison qui ne respecte pas la casse, mais exige néanmoins une correspondance exacte avec le contenu de la barre de titre. Il parcourt d'abord les fenêtres de niveau supérieur, puis les fenêtres enfants. S'il ne détecte aucune correspondance, il lève une exception **ArgumentException**.

Vous ne pouvez utiliser **AppActivate** qu'avec des processus contenant des fenêtres. La plupart des applications console ne possèdent pas de fenêtres, ce qui signifie qu'elles n'apparaissent pas dans la liste de processus que recherche **AppActivate**. Lors de l'exécution à partir d'une application console, le système crée un processus distinct pour exécuter l'application et retourne le résultat au processus console. Par conséquent, lorsque vous demandez l'ID de processus actuel, vous obtenez l'ID de processus de ce processus distinct, plutôt que celui de l'application console.

Au moment de l'exécution, la fonction **AppActivate** active toute application en cours possédant un titre qui correspond à *Title* ou un ID de processus qui correspond à *ProcessId*. S'il n'existe aucune correspondance exacte, elle active toute application dont la chaîne de titre se termine par *Title*. S'il existe plusieurs applications nommées *Title*, la fonction **AppActivate** en choisit arbitrairement une pour l'activer.

Remarque

La fonction **AppActivate** nécessite **UIPermission** au niveau SafeTopLevelWindows, ce qui peut affecter son exécution dans l es situations où le niveau de confiance n'est pas total. Pour plus d'informations, consultez Demande d'autorisations et UIPermission Class.

Exemple

Cet exemple montre comment utiliser la fonction **AppActivate** pour activer une fenêtre d'application. Si un processus Notepad n'est pas en cours d'exécution, l'exemple lève un **ArgumentException**. La procédure **Shell** suppose que les

applications se trouvent dans les chemins d'accès spécifiés.

VB

```
Dim notepadID As Integer
' Activate a running Notepad process.
AppActivate("Untitled - Notepad")
' AppActivate can also use the return value of the Shell function.
' Shell runs a new instance of Notepad.
notepadID = Shell("C:\WINNT\NOTEPAD.EXE", AppWinStyle.NormalFocus)
' Activate the new instance of Notepad.
AppActivate(notepadID)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Shell, fonction

Focus

ArgumentException

Fonctions Asc, AscW

Retourne une valeur de type **Integer** qui représente le code de caractère correspondant à un caractère.

```
Public Overloads Function Asc(ByVal String As Char) As Integer
Public Overloads Function AscW(ByVal String As Char) As Integer
' -or-
Public Overloads Function Asc(ByVal String As String) As Integer
Public Overloads Function AscW(ByVal String As String) As Integer
```

Paramètres

String

Requis. Toute expression **Char** ou **String** valide. Si *String* est une expression **String**, seul le premier caractère de la chaîne est utilisé pour l'entrée. Si *String* a la valeur **Nothing** ou ne contient aucun caractère, une erreur ArgumentException se produit.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*String* est vide ou a une longueur nulle.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Asc retourne le *point de code* ou le code de caractère du caractère d'entrée. Il peut être compris entre 0 et 255 pour les valeurs du jeu de caractères codé sur un octet (SBCS) et entre -32 768 et 32 767 pour les valeurs du jeu de caractères codé sur deux octets (DBCS). Pour les tableaux des caractères ASCII codés sur un octet, consultez Codes de caractères ASCII.

La valeur retournée dépend de la page de codes du thread en cours, contenue dans la propriété ANSICodePage de la classe TextInfo de l'espace de noms System.Globalization. Vous pouvez obtenir **ANSICodePage** en spécifiant `System.Globalization.CultureInfo.CurrentCulture.TextInfo.ANSICodePage`.

AscW retourne le point de code Unicode du caractère d'entrée. Il peut être compris entre 0 et 65 535. La valeur retournée est indépendante des paramètres de la culture et de la page de codes pour le thread en cours.

Remarque

Dans les précédentes versions de Visual Basic, la fonction **AscB** retourne un code pour un octet, plutôt qu'un caractère. Son r ôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Charac ter Set). Toutes les chaînes Visual Basic 2005 sont en Unicode et la fonction **AscB** n'est plus prise en charge.

Exemple

L'exemple suivant utilise la fonction **Asc** pour retourner des codes de caractère Integer, type de données (Visual Basic) correspondant à la première lettre de chaque chaîne.

VB

```
Dim codeInt As Integer
' The following line of code sets myInt to 65.
codeInt = Asc("A")
' The following line of code sets myInt to 97.
codeInt = Asc("a")
' The following line of code sets myInt to 65.
codeInt = Asc("Apple")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Fonctions Chr, ChrW
Fonctions de conversion (Visual Basic)
Fonctions de conversion de types de données
Integer, type de données (Visual Basic)
System.Globalization
CultureInfo
ANSICodePage
ArgumentException

Autres ressources

Codes de caractères ASCII

Beep, fonction

Émet un signal sonore par le haut-parleur de l'ordinateur.

```
Public Sub Beep()
```

Notes

La hauteur et la durée du signal sonore dépendent des caractéristiques du matériel et du logiciel système utilisés et sont donc variables d'un ordinateur à l'autre.

Remarque

La fonction **Beep** nécessite **UIPermission** au niveau SafeTopLevelWindows, ce qui peut affecter son exécution dans les situations où le niveau de confiance n'est pas total. Pour plus d'informations, consultez Demande d'autorisations et UIPermission Class.

Exemple

Cet exemple utilise la fonction **Beep** pour diffuser un signal sonore long et ininterrompu par le haut-parleur de l'ordinateur.

VB

```
Dim I As Integer
For I = 1 To 100   ' Loop 100 times.
   Beep   ' Sound a tone.
Next I
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de la bibliothèque runtime Visual Basic

Autres ressources

Lecture de sons

CallByName, fonction

Applique une méthode à un objet, ou définit ou retourne une propriété d'un objet.

```
Public Function CallByName( _
   ByVal ObjectRef As System.Object, _
   ByVal ProcName As String, _
   ByVal UseCallType As CallType, _
   ByVal Args() As Object _
) As Object
```

Paramètres

ObjectRef

Requis. **Object**. Pointeur qui désigne l'objet exposant la propriété ou la méthode.

ProcName

Requis. **String**. Expression de type chaîne contenant le nom de la propriété ou de la méthode sur l'objet.

UseCallType

Requis. Membre de l'énumération de type CallType représentant le type de procédure appelée. La valeur de **CallType** peut être **Method**, **Get** ou **Set**.

Args

Facultatif. **ParamArray**. Tableau de paramètres contenant les arguments à passer à la propriété ou la méthode appelée.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Valeur *UseCallType* non valide ; doit être **Method**, **Get** ou **Set**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **CallByName** est utilisée lors de l'exécution pour obtenir une propriété, définir une propriété ou appeler une méthode.

Exemple

Dans l'exemple suivant, la première ligne utilise **CallByName** pour définir la propriété **Text** d'une zone de texte, la deuxième ligne récupère la valeur de la propriété **Text** et la troisième ligne appelle la méthode **Move** pour déplacer la zone de texte.

VB

```
' Imports statements must be at the top of a module.
Imports Microsoft.VisualBasic.CallType
```

VB

```
Sub TestCallByName1()
    'Set a property.
    CallByName(TextBox1, "Text", CallType.Set, "New Text")

    'Retrieve the value of a property.
    MsgBox(CallByName(TextBox1, "Text", CallType.Get))

    'Call a method.
    CallByName(TextBox1, "Hide", CallType.Method)
```

```
End Sub
```

L'exemple suivant utilise la fonction **CallByName** pour appeler les méthodes **Add** et **Item** d'un objet de collection.

VB

```
Public Sub TestCallByName2()
    Dim col As New Collection()

    'Store the string "Item One" in a collection by
    'calling the Add method.
    CallByName(col, "Add", CallType.Method, "Item One")

    'Retrieve the first entry from the collection using the
    'Item property and display it using MsgBox().
    MsgBox(CallByName(col, "Item", CallType.Get, 1))
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

CallType, énumération

ArgumentException Class

Concepts

Tableaux de paramètres

Appel d'une propriété ou méthode à l'aide d'un nom de chaîne

ChDir, fonction

Change le répertoire ou le dossier en cours.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que la fonction **ChDir** . Pour plus d'informations, consultez My.Computer.FileSystem.CurrentDirectory, propriété.

```
Public Sub ChDir(ByVal Path As String)
```

Paramètres
Path

Requis. Expression **String** qui identifie le répertoire ou le dossier appelé à devenir le nouveau répertoire ou dossier par défaut. *Path* peut inclure le lecteur. Si aucun lecteur n'est spécifié, la fonction **ChDir** change le répertoire ou le dossier par défaut sur le lecteur actif.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	52	*Path* est vide.
FileNotFoundException	76	Un lecteur non valide est spécifié ou le lecteur est indisponible.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **ChDir** change le répertoire par défaut, mais pas le lecteur par défaut. Par exemple, si le lecteur par défaut est C, l'instruction suivante change le répertoire par défaut sur le lecteur D, mais C reste le lecteur par défaut :

VB

```
ChDir("D:\TMP")
```

L'exécution de changements relatifs de répertoire s'effectue en tapant deux points, comme suit :

VB

```
ChDir("..") ' Moves up one directory.
```

Remarque de sécurité

La fonction **ChDir** nécessite une autorisation de code non managée, qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission, classe et Autorisations d'accès du code.

Exemple

Cet exemple utilise la fonction **ChDir** pour changer le répertoire ou le dossier en cours.

VB

```
' Change current directory or folder to "MYDIR".
ChDir("MYDIR")

' Assume "C:" is the current drive. The following statement changes
' the default directory on drive "D:". "C:" remains the current drive.
ChDir("D:\WINDOWS\SYSTEM")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

ChDrive, fonction
CurDir, fonction
Dir, fonction
MkDir, fonction
RmDir, fonction
ArgumentException Class
FileNotFoundException Class

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

ChDrive, fonction

Change le lecteur actif.

```
Public Overloads Sub ChDrive(ByVal Drive As { Char | String })
```

Paramètres
Drive

Requis. Expression String spécifiant un lecteur existant. Si vous fournissez une chaîne de longueur nulle (""), le lecteur actif ne change pas. Si l'argument *Drive* est une chaîne à plusieurs caractères, la fonction **ChDrive** utilise seulement la première lettre.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	68	Un lecteur non valide est spécifié ou le lecteur est indisponible.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **ChDrive** nécessite une autorisation de code non managée, qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez la classe SecurityPermission et Autorisations d'accès du code.

Exemple

Cet exemple utilise la fonction **ChDrive** pour changer le lecteur actif. La fonction lève une exception si le lecteur n'existe pas.

VB

```
ChDrive("D")   ' Make "D" the current drive.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
ChDir, fonction
CurDir, fonction
MkDir, fonction
RmDir, fonction
IOException Class
Autres ressources
Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Choose, fonction

Sélectionne et retourne une valeur à partir d'une liste d'arguments.

```
Public Function Choose( _
   ByVal Index As Double, _
   ByVal ParamArray Choice() As Object _
) As Object
```

Paramètres

Index

Requis. **Double**. Expression numérique qui produit une valeur comprise entre 1 et le nombre des éléments passés dans l'argument *Choice*.

Choice

Requis. Tableau de paramètres **Object**. Vous pouvez fournir une variable ou une expression unique qui correspond au type de données **Object**, à une liste de variables ou d'expressions **Object** séparées par des virgules ou à un tableau unidimensionnel d'éléments **Object**.

Notes

La fonction **Choose** retourne un membre de la liste passée dans *Choice()*, selon la valeur de *Index*. Le premier membre de la liste est sélectionné lorsque *Index* est 1. Le dernier membre de la liste est sélectionné lorsque *Index* est **UBound**(*Choice()*). Si *Index* se situe en dehors de ces limites, **Choose** retourne **Nothing**.

Si *Index* n'est pas un nombre entier, il est arrondi au nombre entier le plus proche avant d'être évalué.

Vous pouvez utiliser la fonction **Choose** pour examiner une valeur à partir d'une liste de valeurs possibles.

Remarque

Les expressions dans la liste d'arguments peuvent inclure des appels de fonction. Pour préparer la liste d'arguments pour l'appel à **Choose**, le compilateur Visual Basic appelle chaque fonction de chaque expression. Ceci signifie qu'une fonction particulière sera appelée si une expression distincte est sélectionnée par *Index*.

Exemple

Cet exemple utilise la fonction **Choose** pour afficher un nom en réponse à un index passé à la procédure dans le paramètre `Ind`.

VB

```
Function GetChoice(ByVal Ind As Integer) As String
    GetChoice = CStr(Choose(Ind, "Speedy", "United", "Federal"))
End Function
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

IIf, fonction
Select...Case, instruction (Visual Basic)
Switch, fonction

Fonctions Chr, ChrW

Retourne le caractère associé au code de caractère spécifié.

```
Public Function Chr(ByVal CharCode As Integer) As Char
Public Function ChrW(ByVal CharCode As Integer) As Char
```

Paramètres

CharCode

Requis. Expression **Integer** représentant le *point de code* (ou code de caractère) du caractère. Si *CharCode* se trouve en dehors de la plage valide, une erreur ArgumentException se produit. La plage valide pour **Chr** s'étend de 0 à 255, et la plage valide pour **ChrW** s'étend de -32 768 à 65 535.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*CharCode* est inférieur à -32 768 ou supérieur à 65 535 pour **ChrW**.
ArgumentException	5	*CharCode* est inférieur à 0 ou supérieur à 255 pour **Chr**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La plage asymétrique autorisée pour *CharCode* compense les différences de stockage entre les types de données Short, type de données (Visual Basic) et Integer, type de données (Visual Basic). Par exemple, -29 183 est un **Short**, mais +36 353 est un **Integer**. Ceci simplifie également la compatibilité avec Visual Basic 6.0.

Chr utilise la classe Encoding dans l'espace de noms System.Text pour déterminer si le thread en cours utilise un jeu de caractères codé sur un octet (SBCS, Single-Byte Character Set) ou sur deux octets (DBCS, Double-Byte Character Set). Il accepte ensuite *CharCode* comme un point de code dans le jeu approprié. La plage peut s'étendre de 0 à 255 pour les caractères SBCS et de -32 768 à 65 535 pour les caractères DBCS. Pour les tableaux des caractères ASCII codés sur un octet, consultez Codes de caractères ASCII.

La valeur retournée dépend de la page de codes du thread en cours, contenue dans la propriété ANSICodePage de la classe TextInfo de l'espace de noms System.Globalization. Vous pouvez obtenir **ANSICodePage** en spécifiant `System.Globalization.CultureInfo.CurrentCulture.TextInfo.ANSICodePage`.

ChrW accepte *CharCode* comme un point de code Unicode. La plage est indépendante des paramètres de la culture et de la page de code pour le thread en cours. Les valeurs comprises entre -32 768 et -1 sont traitées de la même façon que les valeurs comprises entre +32 768 et +65 535.

Les nombres de 0 à 31 sont identiques aux codes ASCII standard non imprimables. Par exemple, `Chr(10)` retourne un caractère de saut de ligne.

Remarque

La fonction **ChrB** dans les versions antérieures de Visual Basic retourne un octet simple. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Character Set). Toutes les chaînes Visual Basic et .NET Framework sont en Unicode, et la fonction **ChrB** n'est plus prise en charge.

Exemple

L'exemple suivant utilise la fonction **Chr** pour retourner le caractère associé au code spécifié.

VB

```
Dim associatedChar As Char
' Returns "A".
```

```
associatedChar = Chr(65)
' Returns "a".
associatedChar = Chr(97)
' Returns ">".
associatedChar = Chr(62)
' Returns "%".
associatedChar = Chr(37)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Fonctions Asc, AscW
Str, fonction
Fonctions de conversion (Visual Basic)
Fonctions de conversion de types de données
CultureInfo
ArgumentException

Autres ressources

Codes de caractères ASCII

Command, fonction

Retourne la partie argument de la ligne de commande utilisée pour démarrer Visual Basic ou un programme exécutable développé avec Visual Basic.

La fonctionnalité **My** fournit une productivité et une performance supérieures à celles de la fonction **Command** . Pour plus d'informations, consultez My.Application.CommandLineArgs, propriété.

```
Public Function Command() As String
```

Notes

Une fois les arguments retournés, vous pouvez rechercher des délimiteurs communs, tels que les espaces, les barres obliques ou barres obliques inverses, les traits d'union ou les guillemets, pour fractionner ou rechercher les paramètres individuels dans la chaîne.

Pour les applications développées avec Visual Basic et compilées en un fichier .exe, la fonction **Command** retourne tout argument apparaissant après le nom de l'application sur la ligne de commande, comme dans cet exemple : `MyApp(cmdlineargs).`

Exemple

Cet exemple utilise la fonction **Command** pour retourner les arguments de la ligne de commande dans un objet contenant un tableau.

VB

```
Function GetCommandLineArgs() As String()
    ' Declare variables.
    Dim separators As String = " "
    Dim commands As String = Microsoft.VisualBasic.Interaction.Command()
    Dim args() As String = commands.Split(separators.ToCharArray)
    Return args
End Function
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de la bibliothèque runtime Visual Basic
Structure d'un programme Visual Basic
Environment.GetCommandLineArgs Method
My.Application.CommandLineArgs, propriété

Concepts

Version Visual Basic de Hello World!

Autres ressources

Compilateur Visual Basic

CreateObject, fonction (Visual Basic)

Crée et retourne une référence à l'objet COM. **CreateObject** ne peut être utilisé pour créer des instances de classes dans Visual Basic à moins que ces classes soient exposées explicitement en tant que composants COM.

```
Public Shared Function CreateObject( _
   ByVal ProgId As String, _
   Optional ByVal ServerName As String = "" _
) As Object
```

Paramètres

ProgId

Requis. **String**. L'ID de programme de l'objet à créer.

ServerName

Facultatif. **String**. Nom du serveur réseau au niveau duquel l'objet sera créé. Si *ServerName* est une chaîne vide (""), l'ordinateur local est utilisé.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
Exception	429	*ProgId* est introuvable ou non fourni - ou - *ServerName* fait échouer la fonction DnsValidateName très probablement parce qu'elle comprend plus de 63 caractères ou contient un caractère non valide.
Exception	462	Le serveur est indisponible.
FileNotFoundException	53	Aucun objet du type spécifié n'existe

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Pour créer une instance d'un composant COM, assignez l'objet retourné par la fonction **CreateObject** à une variable objet :

```
Sub CreateADODB()
   Dim adoApp As Object
   adoApp = CreateObject("ADODB.Connection")
End Sub
```

Le type de la variable objet que vous utilisez pour stocker l'objet retourné peut affecter les performances de vos applications. La déclaration d'une variable objet avec la clause **As Object** crée une variable pouvant contenir une référence à n'importe quel type d'objet. Cependant, l'accès à l'objet par l'intermédiaire de cette variable est *à liaison tardive*, c'est-à-dire que la liaison est créée lors de l'exécution de votre programme. Vous devez éviter la liaison tardive pour plusieurs raisons, parmi lesquelles le ralentissement de l'exécution de l'application.

Vous pouvez créer une variable objet qui produit une liaison anticipée, c'est-à-dire que la liaison est créée lors de la compilation du programme. Pour cela, ajoutez une référence à la bibliothèque de types pour votre objet à partir de l'onglet **COM** de la boîte de dialogue **Ajouter une référence** dans le menu **Projet**. Puis déclarez la variable objet du type spécifique de votre objet. Dans la plupart des cas, il est plus efficace d'utiliser l'instruction **Dim**et un assembly PIA (Primary Interop Assembly) pour créer des objets plutôt qu'utiliser la fonction **CreateObject**.

Interaction avec le code non managé

Un autre problème est que les objets COM utilisent un code non managé, un code sans l'avantage du Common Language Runtime. Le mélange de code managé Visual Basic et de code non managé COM implique un certain degré de complexité. Quand vous ajoutez une référence à un objet COM, Visual Basic cherche un assembly PIA prédéfini pour cette bibliothèque ; si un assembly est trouvé, il est alors utilisé. Si aucun assembly n'est trouvé, Visual Basic crée un assembly d'interopérabilité qui contient les classes d'interopérabilité locales pour chaque classe de la bibliothèque COM. Pour plus d'informations, consultez Interopérabilité COM dans les applications .NET Framework.

En général, il est préférable d'utiliser des objets étroitement liés et des assemblys PIA (Primary Interop Assembly) dès que cela est possible. Les exemples ci-dessous utilisent la fonction **CreateObject** avec des objets Microsoft Office uniquement dans un but démonstratif. Cependant, ces objets sont plus faciles à utiliser et plus fiables lorsqu'ils sont utilisés avec l'assembly PIA approprié.

Création d'un objet sur un ordinateur distant

Vous pouvez créer un objet sur un ordinateur réseau distant en passant le nom de l'ordinateur à l'argument *ServerName* de la fonction **CreateObject**. Ce nom est identique à une partie du nom de l'ordinateur partagé : pour un partage nommé "\\MyServer\Public", *ServerName*, le nom est "MyServer."

Remarque

Consultez la documentation COM (voir Microsoft Developer Network) pour plus d'informations sur la manière de rendre acc essible une application à partir d'un ordinateur réseau distant. Il vous faudra peut-être ajouter une clé de Registre pour votre application.

Le code suivant retourne le numéro de version d'une instance d'Excel en cours d'exécution sur un ordinateur distant appelé `MyServer` :

```
Sub CreateRemoteExcelObj()
    Dim xlApp As Object
    ' Replace string "\\MyServer" with name of the remote computer.
    xlApp = CreateObject("Excel.Application", "\\MyServer")
    MsgBox(xlApp.Version)
End Sub
```

Si le nom du serveur distant est incorrect, ou si le serveur n'est pas disponible, une erreur se produit au moment de l'exécution.

Remarque

Utilisez **CreateObject** lorsqu'il n'existe aucune instance en cours de l'objet. S'il en existe une, une nouvelle instance est lancé e et un objet du type spécifié est créé. Pour utiliser l'instance en cours ou pour lancer l'application en chargeant un fichier, uti lisez la fonction **GetObject**. Si un objet s'est inscrit comme un objet à instance unique, une seule instance de l'objet est créée, quel que soit le nombre d'exécutions de l'objet **CreateObject**.

Création d'objets Framework

Vous pouvez utiliser la fonction **CreateObject** uniquement pour créer un objet COM. Bien qu'il n'y ait aucun mécanisme équivalent exact pour créer un objet .NET Framework, le Activator dans l'espace de noms System contient des méthodes pour créer des objets locaux ou distants. En particulier, la méthode CreateInstance ou la méthode CreateInstanceFrom peuvent être utiles.

Remarque de sécurité

La fonction **CreateObject** nécessite une permission de code non managée, qui peut affecter son exécution dans les situation s d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

L'exemple suivant utilise la fonction **CreateObject** pour créer une feuille de calcul Microsoft Excel et enregistre la feuille de calcul dans un fichier. Pour utiliser cet exemple, Excel doit être installé sur l'ordinateur où ce programme s'exécute. Vous devez également ajouter une référence à la bibliothèque de types à partir de l'onglet **COM** de la boîte de dialogue **Ajouter une référence** dans le menu **Projet**. Le nom de la bibliothèque de types varie en fonction de la version d'Excel installée sur votre ordinateur. Par exemple, la bibliothèque de types pour Microsoft Excel 2002 se nomme **Microsoft Excel 10.0 Object Library**.

VB

```
Sub TestExcel()
    Dim xlApp As Microsoft.Office.Interop.Excel.Application
    Dim xlBook As Microsoft.Office.Interop.Excel.Workbook
    Dim xlSheet As Microsoft.Office.Interop.Excel.Worksheet

    xlApp = CType(CreateObject("Excel.Application"), _
                Microsoft.Office.Interop.Excel.Application)
    xlBook = CType(xlApp.Workbooks.Add, _
                Microsoft.Office.Interop.Excel.Workbook)
    xlSheet = CType(xlBook.Worksheets(1), _
                Microsoft.Office.Interop.Excel.Worksheet)

    ' The following statement puts text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
    ' The following statement shows the sheet.
    xlSheet.Application.Visible = True
    ' The following statement saves the sheet to the C:\Test.xls directory.
    xlSheet.SaveAs("C:\Test.xls")
    ' Optionally, you can call xlApp.Quit to close the workbook.
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

GetObject, fonction (Visual Basic)
Dim, instruction (Visual Basic)
Declare, instruction
Exception
FileNotFoundException
Activator
CreateInstance
CreateInstanceFrom

Autres ressources

Interopérabilité COM dans les applications .NET Framework
Interopération avec du code non managé

CType, fonction

Retourne le résultat d'une conversion explicite d'une expression en un type de données, objet, structure, classe ou interface spécifié.

```
CType(expression, typename)
```

Éléments

expression

Toute expression valide. Si la valeur de *expression* se trouve en dehors de la plage autorisée par *typename*, Visual Basic lève une exception.

typename

Toute expression valide au sein d'une clause **As** d'une instruction **Dim**, c'est-à-dire le nom de tout type de données, objet, structure, classe ou interface.

Notes

CType est compilé "inline", c'est-à-dire que le code de conversion fait partie du code qui évalue l'expression. Il arrive qu'une conversion soit réalisée sans aucun appel à une procédure, rendant ainsi l'exécution plus rapide.

Si aucune conversion n'est définie de *expression* à *typename*, par exemple de **Integer** à **Date**, Visual Basic affiche un message d'erreur de compilation.

Si une conversion échoue au moment de l'exécution, l'exception appropriée se produit. Si une conversion restrictive échoue, le résultat le plus courant est un OverflowException. Si la conversion n'est pas définie, un InvalidCastException se produit. Par exemple, cela peut arriver si *expression* est de type **Object** et que son type d'exécution n'a aucune conversion en *typename*.

Si le type de données de *expression* ou *typename* est une classe ou structure que vous avez définie, vous pouvez définir **CType** comme opérateur de conversion sur cette classe ou structure. Ainsi, **CType** agit comme un *opérateur surchargé*. En faisant ceci, vous pouvez contrôler le comportement des conversions effectuées vers et à partir de votre classe ou structure, y compris les exceptions pouvant être levées.

Surcharge

L'opérateur **CType** peut également être surchargé sur une classe ou structure définie en dehors de votre code. Si votre code convertit vers et à partir d'une telle classe ou structure, assurez-vous d'avoir bien compris le comportement de son opérateur **CType**. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise la fonction **CType** pour convertir une expression en un type de données spécifié.

VB

```
Dim testNumber As Long = 1000
' The following line of code sets testNewType to 1000.0.
Dim testNewType As Single = CType(testNumber, Single)
```

Voir aussi

Tâches

Comment : définir un opérateur de conversion

Référence

Fonctions de conversion de types de données

Fonctions de conversion (Visual Basic)

Operator, instruction

OverflowException

InvalidCastException

CurDir, fonction

Retourne une chaîne représentant le chemin d'accès en cours.

L'objet My.Computer.FileSystem permet une meilleure productivité et de meilleures performances dans les opérations d'E/S de fichier que **CurDir**. Pour plus d'informations, consultez My.Computer.FileSystem.CurrentDirectory, propriété.

```
Public Overloads Function CurDir([ ByVal Drive As Char ]) As String
```

Paramètres

Drive

Facultatif. Expression **Char** spécifiant un lecteur existant. Si aucun lecteur n'est spécifié ou si *Drive* est une chaîne de longueur nulle (""), **CurDir** retourne le chemin d'accès du lecteur actif.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	68	*Drive* est introuvable.
ArgumentException	68	*Drive* non valide spécifié.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Cette fonction retourne une chaîne représentant le chemin d'accès en cours.

Exemple

Cet exemple utilise la fonction **CurDir** pour retourner le chemin d'accès en cours.

VB

```
' Assume current path on C drive is "C:\WINDOWS\SYSTEM".
' Assume current path on D drive is "D:\EXCEL".
' Assume C is the current drive.
Dim MyPath As String
MyPath = CurDir()   ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("C"c)   ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("D"c)   ' Returns "D:\EXCEL".
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

ChDir, fonction
ChDrive, fonction
MkDir, fonction
RmDir, fonction
IOException Class

ArgumentException Class

Fonctions D-G

Le tableau suivant contient une liste des fonctions membres runtime Visual Basic.

DateAdd	DateDiff	DatePart	DateSerial
DateValue	Day	DDB	DeleteSetting
Derived Math	Dir	Environ	EOF
ErrorToString	FileAttr	FileClose	FileCopy
FileDateTime	FileGet	FileGetObject	FileLen
FileOpen	FilePut	FilePutObject	FileWidth
Filter	Fix	Format	FormatCurrency
FormatDateTime	FormatNumber	FormatPercent	FreeFile
FV	GetAllSettings	GetAttr	GetChar
GetException	GetObject	GetSetting	

La fonction mathématique **Exp**, qui était disponible en tant que fonction membre runtime dans les versions 6.0 et antérieures de Visual Basic, a été remplacée par une méthode (Exp) de la classe Math de la bibliothèque de classes .NET Framework. Pour plus d'informations, consultez Fonctions mathématiques (Visual Basic).

Notes du développeur sur Smart Device

Les éléments suivants ne sont pas pris en charge dans les applications Smart Device : **DeleteSetting**, **Dir**, **Environ**, **EOF**, **FileAttr**, **FileClose**, **FileCopy**, **FileDateTime**, **FileGet**, **FileGetObject**, **FileLen**, **FileOpen**, **FilePut**, **FilePutObject**, **FileWidth**, **FreeFile**, **GetAllSettings**, **GetAttr**, **GetObject** et **GetSetting**.

Voir aussi

Référence

Fonctions A-C
Fonctions H-L
Fonctions M-R
Fonctions S-Z

Autres ressources

Référence Visual Basic

DateAdd, fonction (Visual Basic)

Retourne une valeur **Date** contenant une valeur de date et d'heure auxquelles un intervalle de temps spécifié a été ajouté.

```
Public Overloads Function DateAdd( _
   ByVal Interval As DateInterval, _
   ByVal Number As Double, _
   ByVal DateValue As DateTime _
) As DateTime
' -or-
Public Overloads Function DateAdd( _
   ByVal Interval As String, _
   ByVal Number As Double, _
   ByVal DateValue As Object _
) As DateTime
```

Paramètres

Interval

Requis. Valeur d'énumération **DateInterval** ou expression **String** représentant l'intervalle de temps que vous souhaitez ajouter.

Number

Requis. **Double**. Expression à virgule flottante représentant le nombre d'intervalles à ajouter. *Number* peut être positif (pour obtenir des valeurs date/heure à venir) ou négatif (pour représenter des valeurs date/heures passées). Toute partie fractionnaire de *Number* est ignorée.

DateValue

Requis. **Date**. Expression qui représente la date et l'heure à laquelle l'intervalle sera ajouté. *DateValue* lui-même n'est pas modifié dans le programme appelant.

Paramètres

L'argument *Interval* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Chaîne	Unité de l'intervalle de temps à ajouter
DateInterval.Day	d	Jour ; seule la valeur entière est prise en compte
DateInterval.DayOfYear	y	Jour ; seule la valeur entière est prise en compte
DateInterval.Hour	h	Heure ; seule la valeur entière est prise en compte
DateInterval.Minute	n	Minute ; seule la valeur entière est prise en compte
DateInterval.Month	m	Mois ; seule la valeur entière est prise en compte
DateInterval.Quarter	q	Trimestre ; seule la valeur entière est prise en compte
DateInterval.Second	s	Seconde ; seule la valeur entière est prise en compte
DateInterval.Weekday	w	Jour ; seule la valeur entière est prise en compte
DateInterval.WeekOfYear	ww	Semaine ; seule la valeur entière est prise en compte
DateInterval.Year	yyyy	Année ; seule la valeur entière est prise en compte

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	13	*DateValue* ne peut pas être converti en **Date**.
ArgumentException	5	Interval n'est pas valide.
ArgumentOutOfRangeException	9	La date calculée est antérieure à 00:00:00 le 1er janvier de l'année 1 ou ultérieure à 23:59:59 le 31 décembre 9999.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **DateAdd** pour ajouter ou soustraire à une date un intervalle de temps spécifié. Vous pouvez, par exemple, calculer une date distante de 30 jours ou une heure précédant de 45 minutes l'heure actuelle.

Pour ajouter des jours à *DateValue*, vous pouvez utiliser **DateInterval.Day**, **DateInterval.DayOfYear** ou **DateInterval.Weekday**. Ceux-ci sont traités comme équivalents parce que **DayOfYear** et **Weekday** ne sont pas des intervalles de temps explicites.

La fonction **DateAdd** ne retourne jamais de date non valide. Si nécessaire, la partie jour de la date obtenue est ajustée à la baisse sur le dernier jour du mois obtenu de l'année obtenue. Dans l'exemple suivant, un mois est ajouté à la date du 31 janvier :

```
Dim NextMonth As Date = DateAdd(DateInterval.Month, 1, #1/31/1995#)
```

Dans cet exemple, **DateAdd** retourne `#2/28/1995#`, pas `#2/31/1995#`. Si *DateValue* a la valeur `#1/31/1996#`, la date `#2/29/1996#` est retournée, car 1996 est une année bissextile.

Remarque

DateAdd utilise le paramètre de calendrier actuel de la propriété CurrentCulture de la classe CultureInfo dans l'espace de noms System.Globalization. Les valeurs par défaut de **CurrentCulture** dépendent des paramètres du **Panneau de configuration**.

Étant donné que chaque valeur **Date** est prise en charge par une structure DateTime, ses méthodes vous donnent des possibilités supplémentaires pour ajouter des intervalles de temps. Exemple : vous pouvez ajouter à une variable **Date** un nombre décimal de jours, arrondi à la milliseconde la plus proche, de la façon suivante :

```
Dim NextTime As Date = Now        ' Current date and time.
NextTime = NextTime.AddDays(3.4)  ' Increment by 3 2/5 days.
```

Exemple

Cet exemple prend une date et affiche une date correspondante dans le futur selon le nombre de mois spécifié, à l'aide de la fonction **DateAdd**.

VB

```
Dim Msg, Number, StartDate As String   'Declare variables.
Dim Months As Double
Dim SecondDate As Date
Dim IntervalType As DateInterval
IntervalType = DateInterval.Month   ' Specifies months as interval.
StartDate = InputBox("Enter a date")
SecondDate = CDate(StartDate)
Number = InputBox("Enter number of months to add")
Months = Val(Number)
Msg = "New date: " & DateAdd(IntervalType, Months, SecondDate)
MsgBox(Msg)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DateDiff, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
Day, fonction (Visual Basic)
Format, fonction
Now, propriété
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
Type de données date (Visual Basic)
System

DateDiff, fonction (Visual Basic)

Retourne une valeur de type **Long** spécifiant le nombre d'intervalles de temps entre deux valeurs **Date**.

```
Public Overloads Function DateDiff( _
    ByVal Interval As [ DateInterval | String ], _
    ByVal Date1 As DateTime, _
    ByVal Date2 As DateTime, _
    Optional ByVal DayOfWeek As FirstDayOfWeek = FirstDayOfWeek.Sunday, _
    Optional ByVal  WeekOfYear As FirstWeekOfYear = FirstWeekOfYear.Jan1 _
) As Long
```

Paramètres

Interval

Requis. Valeur d'énumération **DateInterval** ou expression **String** représentant l'intervalle de temps que vous souhaitez utiliser comme unité de la différence entre *Date1* et *Date2*.

Date1

Requis. **Date**. La première valeur de date/d'heure que vous souhaitez utiliser dans le calcul.

Date2

Requis. **Date**. La deuxième valeur de date/d'heure que vous souhaitez utiliser dans le calcul.

DayOfWeek

Facultatif. Valeur choisie dans l'énumération **FirstDayOfWeek** qui spécifie le premier jour de la semaine. Si elle n'est pas spécifiée, **FirstDayOfWeek.Sunday** est utilisé.

WeekOfYear

Facultatif. Valeur choisie dans l'énumération **FirstWeekOfYear** qui spécifie la première semaine de l'année. Si elle n'est pas spécifiée, **FirstWeekOfYear.Jan1** est utilisé.

Paramètres

L'argument *Interval* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur de chaîne	Unité de la différence de temps
DateInterval.Day	"d"	Jour
DateInterval.DayOfYear	"y"	Jour
DateInterval.Hour	"h"	Heure
DateInterval.Minute	"n"	Minute
DateInterval.Month	"m"	Mois
DateInterval.Quarter	"q"	Trimestre
DateInterval.Second	"s"	Seconde
DateInterval.Weekday	"w"	Semaine
DateInterval.WeekOfYear	"ww"	Semaine du calendrier
DateInterval.Year	"yyyy"	Année

L'argument *DayOfWeek* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur	Description
FirstDayOfWeek.System	0	Premier jour de la semaine spécifiée dans les paramètres système
FirstDayOfWeek.Sunday	1	Dimanche (valeur par défaut)
FirstDayOfWeek.Monday	2	Lundi (conformément à la norme ISO 8601, section 3.17)
FirstDayOfWeek.Tuesday	3	Mardi
FirstDayOfWeek.Wednesday	4	Mercredi
FirstDayOfWeek.Thursday	5	Jeudi
FirstDayOfWeek.Friday	6	Vendredi
FirstDayOfWeek.Saturday	7	Samedi

L'argument *WeekOfYear* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur	Description
FirstWeekOfYear.System	0	Première semaine de l'année spécifiée dans les paramètres système
FirstWeekOfYear.Jan1	1	Semaine comprenant le 1er janvier (valeur par défaut).
FirstWeekOfYear.FirstFourDays	2	Semaine comprenant au moins 4 jours de la nouvelle année (conformément à la norme ISO 8601, section 3.17)
FirstWeekOfYear.FirstFullWeek	3	Première semaine entière de la nouvelle année

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Interval* non valide.
ArgumentException	5	*Date1*, *Date2*, or *DayofWeek* est hors limites.
InvalidCastException	13	*Date1* ou *Date2* est d'un type non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **DateDiff** pour déterminer le nombre d'intervalles de temps spécifiés entre deux valeurs date/heure. Par exemple, utilisez **DateDiff** pour calculer le nombre de jours entre deux dates ou le nombre de semaines entre la date d'aujourd'hui et la fin de l'année.

Comportement

- **Traitement des paramètres. DateDiff** soustrait la valeur de *Date1* de la valeur de *Date2* pour obtenir la différence. Aucune de ces valeurs n'est modifiée dans le programme appelant.
- **Valeurs de retour** Dans la mesure où *Date1* et *Date2* appartiennent au type de données **Date**, ils contiennent des valeurs de date et d'heure d'une précision de 100 nanosecondes par rapport à l'horloge système. Cependant, **DateDiff**

retourne toujours le nombre d'intervalles de temps sous forme d'une valeur **Long**.

Si *Date1* représente une date et une heure ultérieures à *Date2*, **DateDiff** retourne un nombre négatif.

- **Intervalles estimés en jours.** Si *Interval* a la valeur **DateInterval.DayOfYear**, il est traité de la même façon que **DateInterval.Day**, parce que **DayOfYear** n'est pas une unité appropriée pour un intervalle de temps.

- **Intervalles estimés en semaines.** Si *Interval* a la valeur **DateInterval.WeekOfYear**, la valeur de retour représente le nombre de semaines écoulées entre le premier jour de la semaine contenant *Date1* et le premier jour de la semaine contenant *Date2*. L'exemple ci-dessous illustre les différents résultats obtenus de **DateInterval.Weekday**.

  VB

  ```
  ' The following statements set datTim1 to a Thursday
  ' and datTim2 to the following Tuesday.
  Dim datTim1 As Date = #1/4/2001#
  Dim datTim2 As Date = #1/9/2001#
  ' Assume Sunday is specified as first day of the week.
  Dim wD As Long = DateDiff(DateInterval.Weekday, datTim1, datTim2)
  Dim wY As Long = DateDiff(DateInterval.WeekOfYear, datTim1, datTim2)
  ```

  Dans l'exemple précédent, **DateDiff** retourne 0 à `wD` parce que la différence entre les dates est inférieure à sept jours, mais retourne 1 à `wY` parce qu'il existe une différence de sept jours entre les premiers jours des semaines de calendrier respectives.

Attention
Lorsque la partie d'heure de *Date1* est supérieure à celle de *Date2* et que *Interval* est défini avec la valeur **DateInterval.WeekOfYear**, la fonction **DateDiff** retourne une valeur qui est inférieure de un par rapport à la valeur correcte.

- **Intervalles plus grands.** Si *Interval* a la valeur **DateInterval.Year**, la valeur de retour est calculée exclusivement à partir des parties année de *Date1* et *Date2*. De la même façon, la valeur de retour de **DateInterval.Month** est calculée exclusivement à partir des parties année et des parties mois des arguments et de celles de **DateInterval.Quarter** à partir des trimestres qui contiennent les deux dates.

  Par exemple, en comparant le 31 décembre au 1er janvier de l'année suivante, **DateDiff** retourne 1 pour **DateInterval.Year**, **DateInterval.Quarter** ou **DateInterval.Month**, bien qu'un seul jour se soit écoulé.

- **Autres intervalles.** Étant donné que chaque valeur **Date** est prise en charge par une structure DateTime, ses méthodes vous donnent des possibilités supplémentaires pour déterminer des intervalles de temps. Par exemple, vous pouvez utiliser la méthode **Subtract** dans l'une ou l'autre de ses formes surchargés : System.DateTime.Subtract(System.TimeSpan) soustrait un TimeSpan d'une variable **Date** pour retourner une autre valeur **Date** et System.DateTime.Subtract(System.DateTime) soustrait une valeur **Date** pour retourner un **TimeSpan**. Vous pouvez mesurer la durée d'un processus en millisecondes, comme dans l'exemple suivant.

  VB

  ```
  Dim startTime As Date = Now
  ' Run the process that is to be timed.
  Dim runLength As Global.System.TimeSpan = Now.Subtract(startTime)
  Dim millisecs As Integer = runLength.Milliseconds
  ```

Exemple

Cet exemple utilise la fonction **DateDiff** pour afficher le nombre de jours entre une date donnée et la date du jour.

VB

```
Dim firstDate, msg As String
Dim secondDate As Date
firstDate = InputBox("Enter a date")
secondDate = CDate(firstDate)
```

```
msg = "Days from today: " & DateDiff(DateInterval.Day, Now, secondDate)
MsgBox(msg)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DateAdd, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
Day, fonction (Visual Basic)
Format, fonction
Now, propriété
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
Type de données date (Visual Basic)
DateTime
TimeSpan

DatePart, fonction (Visual Basic)

Retourne une valeur de type **Integer** contenant le composant spécifié d'une valeur **Date** donnée.

```
Public Overloads Function DatePart( _
   ByVal Interval As DateInterval, _
   ByVal DateValue As DateTime, _
   Optional ByVal FirstDayOfWeekValue As FirstDayOfWeek = VbSunday, _
   Optional ByVal FirstWeekOfYearValue As FirstWeekOfYear = VbFirstJan1 _
) As Integer
' -or-
Public Overloads Function DatePart( _
    ByVal Interval As String, _
   ByVal DateValue As Object, _
   Optional ByVal DayOfWeek As FirstDayOfWeek = FirstDayOfWeek.Sunday, _
   Optional ByVal WeekOfYear As FirstWeekOfYear = FirstWeekOfYear.Jan1 _
) As Integer
```

Paramètres

Interval

Requis. Valeur d'énumération **DateInterval** ou expression **String** représentant la partie de la valeur date/heure que vous souhaitez retourner.

DateValue

Requis. Valeur **Date** à évaluer.

FirstDayOfWeekValue, *DayOfWeek*

Facultatif. Valeur choisie dans l'énumération **FirstDayOfWeek** qui spécifie le premier jour de la semaine. S'il n'est pas spécifié, **FirstDayOfWeek.Sunday** est utilisé.

FirstWeekOfYearValue, *WeekOfYear*

Facultatif. Valeur choisie dans l'énumération **FirstWeekOfYear** qui spécifie la première semaine de l'année. S'il n'est pas spécifié, **FirstWeekOfYear.Jan1** est utilisé.

Paramètres

L'argument *Interval* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Chaîne	Partie de la valeur date/heure à retourner
DateInterval.Day	d	Jour du mois (1 à 31)
DateInterval.DayOfYear	y	Jour de l'année (1 à 366)
DateInterval.Hour	h	Heure
DateInterval.Minute	n	Minute
DateInterval.Month	m	Mois
DateInterval.Quarter	q	Trimestre
DateInterval.Second	s	Seconde
DateInterval.Weekday	w	Jour de la semaine (1 à 7)
DateInterval.WeekOfYear	ww	Semaine de l'année (1 à 53)

DateInterval.Year	yyyy	Année

L'argument *FirstDayOfWeekValue* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur	Description
FirstDayOfWeek.System	0	Premier jour de la semaine spécifiée dans les paramètres système
FirstDayOfWeek.Sunday	1	Dimanche (valeur par défaut)
FirstDayOfWeek.Monday	2	Lundi (conformément à la norme ISO 8601, section 3.17)
FirstDayOfWeek.Tuesday	3	Mardi
FirstDayOfWeek.Wednesday	4	Mercredi
FirstDayOfWeek.Thursday	5	Jeudi
FirstDayOfWeek.Friday	6	Vendredi
FirstDayOfWeek.Saturday	7	Samedi

L'argument *FirstWeekOfYearValue* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur	Description
FirstWeekOfYear.System	0	Première semaine de l'année spécifiée dans les paramètres système
FirstWeekOfYear.Jan1	1	Semaine comprenant le 1er janvier (valeur par défaut).
FirstWeekOfYear.FirstFourDays	2	Semaine comprenant au moins 4 jours de la nouvelle année (conformément à la norme ISO 8601, section 3.17)
FirstWeekOfYear.FirstFullWeek	3	Première semaine entière de la nouvelle année

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Interval* n'est pas valide.
InvalidCastException	13	*DateValue* ne peut pas être converti en **Date**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **DatePart** pour évaluer une valeur date/heure et retourner un composant spécifique. Par exemple, utilisez la fonction **DatePart** pour calculer le jour de la semaine ou l'heure en cours.

Si vous choisissez **DateInterval.Weekday** pour l'argument *Interval*, la valeur retournée est cohérente avec les valeurs de l'énumération **FirstDayOfWeek**. Si vous choisissez **DateInterval.WeekOfYear**, **DatePart** utilise les classes Calendar et CultureInfo de l'espace de noms System.Globalization pour déterminer vos paramètres actuels.

L'argument *FirstDayOfWeekValue* affecte des calculs qui utilisent les paramètres *Interval* de **DateInterval.Weekday** et de **DateInterval.WeekOfYear**. L'argument *FirstWeekOfYearValue* affecte des calculs qui spécifient **DateInterval.WeekOfYear**

pour *Interval*.

Étant donné que chaque valeur **Date** est prise en charge par une structure DateTime, ses méthodes vous donnent des possibilités supplémentaires pour récupérer des éléments de date/heure. Vous pouvez, par exemple, obtenir la valeur date complète d'une variable **Date**, la valeur heure étant minuit, comme dans le code suivant :

```
Dim CurrDatTim As Date = Now   ' Current date and time.
Dim LastMidnight As Date = CurrDatTim.Date   ' At midnight.
```

Exemple

Cet exemple prend une date et affiche le trimestre de l'année en question, à l'aide de la fonction **DatePart**.

VB

```
Dim FirstDate, Msg As String   'Declare variables.
Dim SecondDate As Date
FirstDate = InputBox("Enter a date:")
SecondDate = CDate(FirstDate)
Msg = "Quarter: " & DatePart(DateInterval.Quarter, SecondDate)
MsgBox(Msg)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DateAdd, fonction (Visual Basic)
DateDiff, fonction (Visual Basic)
Day, fonction (Visual Basic)
Format, fonction
Now, propriété
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
Type de données date (Visual Basic)
System

DateSerial, fonction (Visual Basic)

Retourne une valeur **Date** représentant une année, un mois et un jour spécifiés, l'indication d'heure ayant la valeur minuit (00:00:00).

```
Public Function DateSerial( _
   ByVal [Year] As Integer, _
   ByVal [Month] As Integer, _
   ByVal [Day] As Integer _
) As DateTime
```

Paramètres

Year

Requis. Expression **Integer** comprise entre 1 et 9999. Cependant, les valeurs inférieures à cette plage sont également acceptées. Si *Year* a une valeur comprise entre 0 et 99, elle est interprétée comme étant située entre 1930 et 2029, conformément à ce qui est expliqué dans la section « Notes » ci-dessous. Si *Year* a une valeur inférieure à 1, cette dernière est soustraite de l'année en cours.

Month

Requis. Expression **Integer** comprise entre 1 et 12. Cependant, les valeurs non comprises dans cette plage sont également acceptées. La valeur de *Month* est décalée de 1 et appliquée au mois de janvier de l'année calculée. En d'autres termes, (*Month* - 1) est ajouté à janvier. L'année est recalculée si nécessaire. Les résultats suivants illustrent cet effet :

- Si *Month* a la valeur 1, le résultat est janvier de l'année calculée.
- Si *Month* a la valeur 0, le résultat est décembre de l'année précédente.
- Si *Month* a la valeur -1, le résultat est novembre de l'année précédente.
- Si *Month* a la valeur 13, le résultat est janvier de l'année suivante.

Day

Requis. Expression **Integer** comprise entre 1 et 31. Cependant, les valeurs non comprises dans cette plage sont également acceptées. La valeur de *Day* est décalée de 1 et appliquée au premier jour du mois calculé. En d'autres termes, (*Day* - 1) est ajouté au premier jour du mois. Le mois et l'année sont recalculés si nécessaire. Les résultats suivants illustrent cet effet :

- Si *Day* a la valeur 1, le résultat est le premier jour du mois calculé.
- Si *Day* a la valeur 0, le résultat est le dernier jour du mois précédent.
- Si *Day* a la valeur -1, le résultat est l'avant-dernier jour du mois précédent.
- Si *Day* est ultérieur à la fin du mois en cours, le résultat est le jour correspondant du mois suivant. Par exemple, si *Month* a la valeur 4 et *Day* la valeur 31, le résultat est le 1er mai.

Notes

Sous Windows 98 ou Windows 2000, les années sur deux chiffres fournies pour l'argument *Year* sont interprétées en fonction des paramètres de l'ordinateur définis par l'utilisateur. Les paramètres par défaut sont les valeurs comprises entre 0 et 29, interprétées comme les années 2000–2029, et celles comprises entre 30 et 99, interprétées comme les années 1930–1999. Pour tous les autres arguments *Year*, exprimez l'année sous la forme d'un nombre à quatre chiffres, par exemple 1924.

Les versions antérieures de Windows interprètent les années à deux chiffres en fonction des paramètres par défaut décrits précédemment. Pour vous assurer que la fonction retourne la valeur correcte, utilisez une année (*Year*) à quatre chiffres.

L'exemple à suivre contient des valeurs d'arguments négatives, nulles et positives. Dans ce cas, la fonction **DateSerial** retourne une **Date** représentant le jour précédant le premier jour du mois de mars de l'année située 10 ans avant l'année en cours, soit en d'autres termes le dernier jour de février il y a dix ans.

```
Dim EndFeb As Date = DateSerial(-10, 3, 0)
```

Si *Month* ou *Day* dépasse la plage normale, la valeur est appliquée à l'unité supérieure pertinente. Par exemple, si vous spécifiez 32 jours, ce chiffre est interprété comme un mois plus entre un et quatre jours, selon la valeur de *Month*. Si *Year* est supérieure à 9999 ou si n'importe quel argument se situe en dehors de la plage allant de -2 147 483 648 à 2 147 483 647, une erreur ArgumentException se produit. Si la date spécifiée par les trois arguments est antérieure à 00:00:00 le 1er janvier de l'année 1 ou ultérieure à 23:59:59 le 31 décembre 9999, une erreur ArgumentOutOfRangeException se produit.

Le type de données **Date** inclut les composants heure. **DateSerial** leur affecte tous la valeur 0, de sorte que la valeur retournée représente le début du jour calculé.

Étant donné que chaque valeur **Date** est prise en charge par une structure DateTime, ses méthodes vous donnent des possibilités supplémentaires pour assembler une valeur **Date**. Par exemple, vous pouvez utiliser l'un des constructeurs DateTime surchargés pour remplir une variable **Date** à l'aide de la combinaison de composants souhaitée. L'exemple suivant affecte à `NewDateTime` la valeur du 6 mai 1978 à un dixième de seconde avant 8 heures 30 :

```
Dim NewDateTime As Date = New Date(1978, 5, 6, 8, 29, 59, 900)
```

Exemple

Cet exemple utilise la fonction **DateSerial** pour retourner la date de l'année, du mois et du jour spécifiés.

VB

```
Dim MyDate As Date
' MyDate contains the date for February 12, 1969.
MyDate = DateSerial(1969, 2, 12)   ' Return a date.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DateValue, fonction (Visual Basic)
Day, fonction (Visual Basic)
Month, fonction (Visual Basic)
Now, propriété
TimeSerial, fonction (Visual Basic)
TimeValue, fonction (Visual Basic)
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
Type de données date (Visual Basic)
System

DateValue, fonction (Visual Basic)

Retourne une valeur **Date** contenant l'indication de date représentée par une chaîne, l'indication d'heure ayant la valeur minuit (00:00:00).

```
Public Function DateValue(ByVal StringDate As String) As DateTime
```

Paramètres
StringDate

Requis. Expression **String** représentant une valeur de date/d'heure située entre 00:00:00 le 1er janvier de l'année 1 et 23:59:59 le 31 décembre 9999.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	13	*StringDate* comprend une indication d'heure non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si *StringDate* comprend uniquement des nombres de 1 à 12 séparés par des séparateurs de date valides, **DateValue** reconnaît l'ordre du mois, du jour et de l'année d'après le format Date courte spécifié pour votre système. **DateValue** utilise le paramètre de calendrier actuel de la propriété CurrentCulture de la classe CultureInfo dans l'espace de noms System.Globalization. Les valeurs par défaut de **CurrentCulture** dépendent des paramètres du **Panneau de configuration**. Vous pouvez substituer le format Date courte en définissant la propriété ShortDatePattern de la classe DateTimeFormatInfo dans l'espace de noms **System.Globalization**.

DateValue reconnaît les noms de mois sous leur forme longue, abrégée et numérique. Par exemple, **DateValue** reconnaît non seulement 30/12/1991 et 30/12/91, mais aussi 30 décembre 1991 et 30-déc-1991.

Si la partie année de l'argument *StringDate* est omise, **DateValue** prend comme valeur l'année en cours fournie par la date.

Si l'argument *StringDate* comprend une indication d'heure, **DateValue** ne l'inclut pas dans la valeur retournée. Si, toutefois, l'heure indiquée dans *StringDate* n'est pas valide (par exemple "89:98"), une erreur **InvalidCastException** se produit.

Exemple

Cet exemple utilise la fonction **DateValue** pour convertir une chaîne en date. Vous pouvez également utiliser des littéraux de date pour assigner directement une date à une variable **Object** ou **Date**, par exemple, `oldDate = #2/12/69#`.

VB

```
Dim oldDate As Date
oldDate = DateValue("February 12, 1969")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
DateSerial, fonction (Visual Basic)
Day, fonction (Visual Basic)
Month, fonction (Visual Basic)
Now, propriété

TimeSerial, fonction (Visual Basic)
TimeValue, fonction (Visual Basic)
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
Type de données date (Visual Basic)
DateTime

Day, fonction (Visual Basic)

Retourne une valeur **Integer** comprise entre 1 et 31 représentant le jour du mois.

```
Public Function Day(ByVal DateValue As DateTime) As Integer
```

Paramètres
DateValue

Requis. Valeur **Date** de laquelle vous souhaitez extraire le jour.

Notes

Si vous utilisez la fonction **Day**, vous devrez peut-être la qualifier à l'aide de l'espace de noms **Microsoft.VisualBasic**, parce que l'espace de noms System.Windows.Forms définit Day comme une énumération. L'exemple suivant montre comment la qualification de `Day` permet de résoudre cette ambiguïté :

```
Dim thisDay As Integer = Microsoft.VisualBasic.DateAndTime.Day(Now)
```

Vous pouvez également obtenir le jour du mois en appelant **DatePart** et en spécifiant `DateInterval.Day` pour l'argument *Interval*.

Exemple

L'exemple suivant utilise la fonction **Day** pour obtenir le jour du mois à partir d'une date spécifiée. Dans l'environnement de développement, le littéral de date est affiché sous le format de date courte standard (par exemple "02/12/1969") en utilisant les paramètres régionaux de votre code.

VB

```
Dim oldDate As Date
Dim oldDay As Integer
' Assign a date using standard short format.
oldDate = #2/12/1969#
oldDay = Microsoft.VisualBasic.DateAndTime.Day(oldDate)
' oldDay now contains 12.
```

La fonction `Day` est qualifiée pour se distinguer de l'énumération **System.Windows.Forms.Day**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
DatePart, fonction (Visual Basic)
Month, fonction (Visual Basic)
Now, propriété
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
DateTime
ArgumentException
ArgumentOutOfRangeException

DDB, fonction

Retourne une valeur **Double** indiquant la valeur d'amortissement d'un bien sur une période déterminée en utilisant la méthode d'amortissement dégressif à taux double ou toute autre méthode spécifiée.

```
Function DDB( _
   ByVal Cost As Double, _
   ByVal Salvage As Double, _
   ByVal Life As Double, _
   ByVal Period As Double, _
   Optional ByVal Factor As Double = 2.0 _
) As Double
```

Paramètres

Cost

Requis. Donnée de type **Double** indiquant le coût initial du bien.

Salvage

Requis. Donnée de type **Double** indiquant la valeur du bien à la fin de son cycle de vie.

Life

Requis. Donnée de type **Double** indiquant la durée du cycle de vie du bien.

Period

Requis. Donnée de type **Double** indiquant la période sur laquelle l'amortissement du bien est calculé.

Factor

Facultatif. Donnée de type **Double** indiquant le taux utilisé pour le calcul de l'amortissement. Si cet argument est omis, la valeur 2 (méthode d'amortissement dégressif à taux double) est utilisée par défaut.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Factor* < = 0, *Salvage* < 0, *Period* < = 0, ou *Period* > *Life*.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La méthode d'amortissement dégressif à taux double calcule l'amortissement à un taux accéléré. L'amortissement est plus important au cours de la première période, puis ralentit au cours des périodes suivantes.

Les arguments *Life* et *Period* doivent être exprimés dans les mêmes unités. Par exemple, si *Life* est donné en mois, *Period* doit être également donné en mois. Tous les arguments doivent avoir pour valeur des nombres positifs.

La fonction **DDB** utilise la formule suivante pour calculer l'amortissement sur une période donnée :

*Depreciation / Period = ((Cost – Salvage) * Factor) / Life*

Exemple

Cet exemple utilise la fonction **DDB** pour retourner la valeur d'amortissement d'un bien sur une période déterminée, en fonction de son coût initial (`InitCost`), de sa valeur résiduelle à l'issue de sa période d'utilisation (`SalvageVal`), de sa durée de vie totale en nombre d'années (`LifeTime`) et de la période en nombre d'années sur laquelle l'amortissement est calculé (`Depr`).

VB

```
Dim InitCost, SalvageVal, LifeTime, DepYear As Double
Dim Fmt As String = "###,##0.00"
```

```
InitCost = CDbl(InputBox("What's the initial cost of the asset?"))
SalvageVal = CDbl(InputBox("Enter the asset's value at end of its life."))
LifeTime = CDbl(InputBox("What's the asset's useful life in years?"))

' Use the SLN function to calculate the deprecation per year.
Dim SlnDepr As Double = SLN(InitCost, SalvageVal, LifeTime)
Dim msg As String = "The depreciation per year: " & Format(SlnDepr, Fmt)
msg &= vbCrLf & "Year" & vbTab & "Linear" & vbTab & "Doubling" & vbCrLf

' Use the SYD and DDB functions to calculate the deprecation for each year.
For DepYear = 1 To LifeTime
    msg &= DepYear & vbTab & _
        Format(SYD(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbTab & _
        Format(DDB(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbCrLf
Next
MsgBox(msg)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

SLN, fonction
SYD, fonction
Liste des mots clés financiers
ArgumentException

DeleteSetting, fonction

Supprime un paramètre de section ou de clé pour une application dans la base de registres de Windows.

La fonctionnalité **My** vous permet une meilleure productivité et de meilleures performances dans les opérations de Registre que la fonction **DeleteSetting** Pour plus d'informations, consultez My.Computer.Registry, objet.

```
Public Sub DeleteSetting( _
   ByVal AppName As String, _
   Optional ByVal Section As String = Nothing, _
   Optional ByVal Key As String = Nothing _
)
```

Paramètres

AppName

Requis. Expression **String** contenant le nom de l'application ou du projet auquel s'applique le paramètre de section ou de clé.

Section

Requis. Expression **String** contenant le nom de la section à partir de laquelle le paramètre de clé est supprimé. Si seuls *AppName* et *Section* sont fournis, la section spécifiée est supprimée avec tous les paramètres de clé connexes.

Key

Facultatif. Expression **String** contenant le nom du paramètre de clé supprimé.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Paramètre *Section*, *AppName* ou *Key* inexistant.
ArgumentException	5	L'utilisateur n'est pas connecté.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si tous les arguments sont fournis, le paramètre spécifié est supprimé. Une erreur d'exécution se produit si vous tentez d'utiliser **DeleteSetting** sur un paramètre de section ou de clé inexistant.

DeleteSetting nécessite qu'un utilisateur soit connecté, car elle fonctionne sous la clé de Registre **HKEY_LOCAL_USER**, qui n'est active que lorsque l'utilisateur se connecte de façon interactive.

Les paramètres de Registre accessibles via un processus non interactif (tel que Mtx.exe) doivent être stockés au-dessous des clés de Registre **HKEY_LOCAL_MACHINE\Software** ou **HKEY_USER\DEFAULT\Software**.

Exemple

L'exemple suivant utilise d'abord la procédure **SaveSetting** pour entrer des données dans la base de registres de Windows pour l'application `MyApp`, puis utilise la fonction **DeleteSetting** pour les supprimer. Dans la mesure où aucun argument *Key* n'est spécifié, l'ensemble de la section `Startup` est supprimé, y compris le nom de la section et toutes ses clés.

VB

```
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp","Startup", "Left", "50")
' Remove section and all its settings from registry.
DeleteSetting ("MyApp", "Startup")
' Remove MyApp from the registry.
DeleteSetting ("MyApp")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms :Microsoft.VisualBasic

Module : **Interaction**

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

GetAllSettings, fonction
GetSetting, fonction
SaveSetting, fonction
ArgumentException Class

Dir, fonction

Retourne une chaîne représentant le nom d'un fichier, d'un répertoire ou d'un dossier qui correspond à un modèle ou un attribut de fichier spécifié ou à l'étiquette de volume d'un lecteur.

My.Computer.FileSystem, objet permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que la fonction **Dir**. Pour plus d'informations, consultez My.Computer.FileSystem.GetDirectoryInfo, méthode.

```
Public Overloads Function Dir() As String
' -or-
Public Overloads Function Dir( _
       ByVal PathName As String, _
       Optional ByVal Attributes As FileAttribute = FileAttribute.Normal _
) As String
```

Paramètres

PathName

Facultatif. Expression **String** qui spécifie un nom de fichier, de répertoire ou de dossier ou un nom de volume de lecteur. Une chaîne de longueur nulle (**""**) est retournée si *PathName* est introuvable.

Attributes

Facultatif. Énumération ou expression numérique dont la valeur spécifie des attributs de fichier. Si cet argument est omis, **Dir** retourne les fichiers qui correspondent à *PathName*, mais qui n'ont pas d'attribut.

Paramètres

Les valeurs d'énumération de l'argument *Attributes* sont les suivantes :

Valeur	Constante	Description
Normal	**vbnormal**	Par défaut. Spécifie les fichiers sans attributs.
ReadOnly	**vbReadOnly**	Spécifie les fichiers en lecture seule en plus des fichiers sans attributs.
Hidden	**vbHidden**	Spécifie les fichiers masqués en plus des fichiers sans attributs.
System	**vbSystem**	Spécifie les fichiers système en plus des fichiers sans attributs.
Volume	**vbVolume**	Spécifie un nom de volume. Si aucun autre attribut n'est spécifié, **vbVolume** est ignoré.
Directory	**vbDirectory**	Spécifie les répertoires ou les dossiers en plus des fichiers sans attributs.
Archive	**vbArchive**	Fichier modifié depuis la dernière sauvegarde.
Alias	**vbAlias**	Le fichier a un nom différent.

Remarque

Ces énumérations spécifiées par le langage Visual Basic peuvent remplacer les valeurs réelles partout dans votre code.

Notes

La fonction **Dir** prend en charge l'utilisation de caractères génériques à plusieurs caractères (*) et à un seul caractère (?) pour spécifier plusieurs fichiers.

VbVolume retourne le nom de volume pour le lecteur au lieu d'un nom de fichier spécifique.

Vous devez fournir un *PathName* la première fois que vous appelez la fonction **Dir**. Pour récupérer l'élément suivant, vous pouvez effectuer les appels suivants à la fonction **Dir** sans paramètres.

Remarque de sécurité

Pour que la fonction **Dir** s'exécute correctement, les indicateurs Read et PathDiscovery de FileIOPermission doivent être accordés au code exécutant. Pour plus d'informations, consultez **FileIOPermission**, SecurityException et Autorisations d'accès du code.

Exemple

Cet exemple utilise la fonction **Dir** pour vérifier si certains fichiers et répertoires existent.

VB

```
Dim MyFile, MyPath, MyName As String
' Returns "WIN.INI" if it exists.
MyFile = Dir("C:\WINDOWS\WIN.INI")

' Returns filename with specified extension. If more than one *.INI
' file exists, the first file found is returned.
MyFile = Dir("C:\WINDOWS\*.INI")

' Call Dir again without arguments to return the next *.INI file in the
' same directory.
MyFile = Dir()

' Return first *.TXT file, including files with a set hidden attribute.
MyFile = Dir("*.TXT", vbHidden)

' Display the names in C:\ that represent directories.
MyPath = "c:\"   ' Set the path.
MyName = Dir(MyPath, vbDirectory)   ' Retrieve the first entry.
Do While MyName <> ""   ' Start the loop.
      ' Use bitwise comparison to make sure MyName is a directory.
      If (GetAttr(MyPath & MyName) And vbDirectory) = vbDirectory Then
         ' Display entry only if it's a directory.
         MsgBox(MyName)
      End If
   MyName = Dir()   ' Get next entry.
Loop
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

ChDir, fonction
CurDir, fonction
FileAttribute, énumération

Environ, fonction

Retourne la chaîne associée à une variable d'environnement de système d'exploitation.

```
Overloads Function Environ(ByVal Expression As Integer) As String
' -or-
Overloads Function Environ(ByVal Expression As String) As String
```

Paramètres

Expression

Requis. Expression qui évalue soit une chaîne contenant le nom d'une variable d'environnement, soit un entier correspondant à l'ordre numérique d'une chaîne d'environnement de la table de chaînes d'environnement.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Expression* est manquante.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si *Expression* contient une chaîne, la fonction Environ retourne le texte assigné à la chaîne d'environnement spécifiée, c'est-à-dire le texte qui suit le signe égal (=) dans la table de chaînes d'environnement pour cette variable d'environnement. Si la chaîne dans *Expression* est introuvable dans la table de chaînes d'environnement, une chaîne de longueur nulle ("") est retournée.

Si *Expression* contient un entier, la chaîne qui occupe cette position numérique dans la table de chaînes d'environnement est retournée. Dans ce cas, la fonction **Environ** retourne la totalité du texte, y compris le nom de la variable d'environnement. Si aucune chaîne d'environnement ne se trouve dans la position spécifiée, la fonction **Environ** retourne une chaîne de longueur nulle.

Remarque de sécurité
La fonction **Environ** nécessite une permission d'environnement, qui peut affecter son exécution dans les situations d'un nive au de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

Cet exemple utilise la fonction **Environ** pour fournir le numéro d'entrée et la longueur de l'instruction `PATH` à partir de la table de chaînes d'environnement.

VB

```
Sub tenv()
   Dim envString As String
   Dim found As Boolean = False
   Dim index As Integer = 1
   Dim pathLength As Integer
   Dim message As String

   envString = Environ(index)
   While Not found And (envString <> "")
      If (envString.Substring(0, 5) = "Path=") Then
         found = True
      Else
         index += 1
         envString = Environ(index)
      End If
   End While
```

```
    If found Then
       pathLength = Environ("PATH").Length
       message = "PATH entry = " & index & " and length = " & pathLength
    Else
       message = "No PATH environment variable exists."
    End If

    MsgBox(message)
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de la bibliothèque runtime Visual Basic

ArgumentException

EOF, fonction

Retourne une valeur Boolean **True** lorsque la fin d'un fichier ouvert en mode **Random** ou **Input** séquentiel est atteinte.

```
Public Function EOF(ByVal FileNumber As Integer) As Boolean
```

Paramètres
FileNumber

Requis. Valeur de type **Integer** contenant tout numéro de fichier valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **EOF** pour éviter de générer une erreur lorsque vous tentez d'obtenir des données au-delà de la fin d'un fichier.

La fonction **EOF** retourne la valeur **False** tant que la fin du fichier n'est pas atteinte. Avec des fichiers ouverts en mode **Random** ou **Binary**, la fonction **EOF** retourne la valeur **False** tant que la dernière fonction **FileGet** exécutée peut lire un enregistrement dans son intégralité.

Dans le cas de fichiers ouverts en mode **Binary**, une erreur se produit si vous tentez de lire le fichier à l'aide de la fonction **Input** jusqu'à ce que la fonction **EOF** retourne la valeur **True**. Utilisez les fonctions **LOF** et **Loc** au lieu de **EOF** pour la lecture de fichiers binaires à l'aide de **Input** ou utilisez **Get** avec la fonction **EOF**. Dans le cas de fichiers ouverts en mode **Output**, **EOF** retourne toujours la valeur **True**.

Exemple

Cet exemple utilise la fonction **EOF** pour détecter la fin d'un fichier. Nous supposons que `Testfile` est un fichier texte comportant quelques lignes de texte.

VB

```
Dim TextLine As String
' Open file.
FileOpen(1, "TESTFILE", OpenMode.Input)
' Loop until end of file.
Do While Not EOF(1)
' Read line into variable.
   TextLine = LineInput(1)
' Display result in a message box.
   MsgBox("End of file reached at " & TextLine)
Loop
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileGet, fonction
Loc, fonction
LOF, fonction
FileOpen, fonction
IOException

ErrorToString, fonction

Retourne le message d'erreur qui correspond à un numéro d'erreur donné.

```
Public Shared Function ErrorToString(ByVal ErrorNumber As Integer) As String
```

Paramètres
ErrorNumber

Facultatif. Tout numéro d'erreur valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*ErrorNumber* est hors limites.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **ErrorToString** examine les paramètres de propriété de l'objet **Err** afin d'identifier l'erreur d'exécution la plus récente. La valeur de retour de la fonction **ErrorToString** correspond à la propriété **Description** de l'objet **Err**. Si *ErrorNumber* est un numéro d'erreur valide, mais n'est pas défini, **ErrorToString** retourne la chaîne `"Application-defined or object-defined error."`. Si *ErrorNumber* n'est pas valide, une erreur se produit. Si *ErrorNumber* est omis, le message correspondant à l'erreur d'exécution la plus récente est retourné. Si aucune erreur d'exécution ne s'est produite ou si *ErrorNumber* est égal à 0, **ErrorToString** retourne une chaîne de longueur nulle ("").

Dans Visual Basic version 6.0 et antérieure, cette fonctionnalité était fournie par la fonction **Error**.

Exemple

Le code suivant utilise la fonction **ErrorToString** pour afficher les messages d'erreur qui correspondent aux numéros d'erreur spécifiés :

VB

```
Dim ErrorNumber As Integer
For ErrorNumber = 61 To 64   ' Loop through values 61 - 64.
   MsgBox(ErrorToString(ErrorNumber))   ' Display error names in message box.
Next ErrorNumber
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence
Err, objet (Visual Basic)
Description, propriété (objet Err)
ArgumentException

Concepts
Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic
Vue d'ensemble de la gestion non structurée des exceptions

FileAttr, fonction

Retourne une énumération représentant le mode d'ouverture des fichiers avec la fonction **FileOpen**.

L'objet My.Computer.FileSystem permet une meilleure productivité et de meilleures performances dans les opérations d'E/S de fichier que la fonction **FileAttr** . Pour plus d'informations, consultez My.Computer.FileSystem.GetFileInfo, méthode.

```
Public Function FileAttr(ByVal FileNumber As Integer) As OpenMode
```

Paramètres
FileNumber

Requis. **Integer** Tout numéro de fichier valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Valeur de retour

Les valeurs d'énumération suivantes indiquent le mode d'accès au fichier :

Valeur	Mode
1	**OpenMode.Input**
2	**OpenMode.Output**
4	**OpenMode.Random**
8	**OpenMode.Append**
32	**OpenMode.Binary**

Notes

Cette fonction retourne une énumération représentant le mode d'ouverture des fichiers à l'aide de la fonction **FileOpen**.

Exemple

Cet exemple utilise la fonction **FileAttr** pour retourner le mode d'accès d'un fichier ouvert.

VB

```
Dim mode As OpenMode
FileOpen(1, "c:\TESTFILE.TXT", OpenMode.Input)
mode = FileAttr(1)
MsgBox("The file mode is " & mode.ToString())
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

GetAttr, fonction
FileOpen, fonction
SetAttr, fonction
IOException Class

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

FileClose, fonction

Termine les opérations d'entrée/sortie (E/S) dans un fichier ouvert à l'aide de la fonction **FileOpen**.

L'utilisation de **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub FileClose(ParamArray FileNumbers() As Integer)
```

Paramètres
FileNumbers

Facultatif. Tableau de paramètres de 0 canaux ou plus à fermer.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **FileClose** est fournie pour la compatibilité descendante et peut affecter la performance. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Si l'argument *FileNumbers* est omis, tous les fichiers actifs ouverts à l'aide de la fonction **FileOpen** sont fermés.

Lorsque vous fermez un fichier ouvert en mode **Output** ou **Append**, la mémoire tampon finale de sortie est écrite dans la mémoire tampon du système d'exploitation associée à ce fichier. La totalité de l'espace dans la mémoire tampon associée au fichier fermé est libérée.

Lorsque la fonction **FileClose** est exécutée, l'association d'un fichier avec son numéro de fichier est désactivée.

Exemple

Cet exemple utilise la fonction **FileClose** pour fermer un fichier ouvert dans le cadre de la fonction **Input**.

VB

```
Dim TextLine As String
FileOpen(1, "TESTFILE", OpenMode.Input)   ' Open file.
Do While Not EOF(1)   ' Loop until end of file.
   TextLine = LineInput(1)   ' Read line into variable.
   MsgBox(TextLine)   ' Display the line
Loop
FileClose(1)   ' Close file.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

End, instruction

FileOpen, fonction

Reset, fonction

Stop, instruction (Visual Basic)

Autres ressources

Accès au fichier avec Visual Basic

FileCopy, fonction

Copie un fichier.

L'objet My.Computer.FileSystem permet une meilleure productivité et de meilleures performances dans les opérations d'E/S de fichier que **FileCopy**. Pour plus d'informations, consultez My.Computer.FileSystem.CopyFile, méthode.

```
Public Sub FileCopy( _
   ByVal Source As String, _
   ByVal Destination As String _
)
```

Paramètres

Source

Requis. Expression **String** qui spécifie le nom du fichier à copier. *Source* peut inclure le répertoire ou le dossier, et le lecteur du fichier source.

Destination

Requis. Expression **String** qui spécifie le nom de fichier cible. *Destination* peut inclure le répertoire ou le dossier, et le lecteur du fichier de destination.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	52	*Source* ou *Destination* n'est pas valide ou n'est pas spécifié.
IOException	55	Le fichier est déjà ouvert
FileNotFoundException	53	Le fichier n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si vous essayez d'utiliser la fonction **FileCopy** sur un fichier ouvert, une erreur se produit.

FileCopy requiert une confiance totale pour utiliser le lecteur local.

Exemple

Cet exemple utilise la fonction **FileCopy** pour copier un fichier vers un autre. Pour cet exemple, supposons que `SrcFile` est un fichier contenant des données.

VB

```
Dim SourceFile, DestinationFile As String
SourceFile = "SRCFILE"    ' Define source file name.
DestinationFile = "DESTFILE"    ' Define target file name.
FileCopy(SourceFile, DestinationFile)    ' Copy source to target.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic

Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic

Comment : copier un répertoire dans un autre répertoire dans Visual Basic

Référence

Kill, fonction

IOException Class

FileNotFoundException Class

ArgumentException

FileDateTime, fonction

Retourne une valeur **Date** qui indique la date et l'heure de création ou de dernière modification d'un fichier.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FileDateTime**. Pour plus d'informations, consultez My.Computer.FileSystem.GetFileInfo, méthode.

```
Public Function FileDateTime(ByVal PathName As String) As DateTime
```

Paramètres
PathName

Requis. Expression **String** qui spécifie un nom de fichier. *PathName* peut comprendre le répertoire ou le dossier, et le lecteur.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	52	*PathName* n'est pas valide ou contient des caractères génériques.
FileNotFoundException	53	Le fichier cible n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Cette fonction retourne une valeur **Date** qui indique la date et l'heure de création ou de dernière modification d'un fichier.

Exemple

Cet exemple utilise la fonction **FileDateTime** pour déterminer la date et l'heure de création ou de dernière modification d'un fichier. Le format de date et d'heure affiché est basé sur les paramètres régionaux de votre système.

VB

```
Dim MyStamp As Date
' Assume TESTFILE was last modified on October 12, 2001 at 4:35:47 PM.
' Assume English/U.S. locale settings.
' Returns "10/12/2001 4:35:47 PM".
MyStamp = FileDateTime("C:\TESTFILE.txt")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
FileLen, fonction
GetAttr, fonction
ArgumentException Class
FileNotFoundException Class
Autres ressources
Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

FileGet, fonction

Lit les données d'un fichier disque ouvert et les place dans une variable.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FileGet**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Object, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Double, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As System.Array, _
```

```
    Optional RecordNumber As Integer = -1, _
    Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Public Overloads Sub FileGet(
    ByVal FileNumber As Integer, _
    ByRef Value As String, _
    Optional RecordNumber As Integer = -1, _
    Optional StringIsFixedLength As Boolean = False _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Value

Requis. Nom de variable valide destiné à recevoir les données lues.

RecordNumber

Facultatif. Numéro de l'enregistrement (fichiers ouverts en mode **Random**) ou de l'octet (fichiers ouverts en mode **Binary**) par lequel la lecture débute.

ArrayIsDynamic

Facultatif. S'applique uniquement à l'écriture d'un tableau. Spécifie si le tableau doit être considéré comme dynamique et si un descripteur de tableau décrivant la taille et les limites du tableau s'avère nécessaire.

StringIsFixedLength

Facultatif. S'applique uniquement lors de l'écriture d'une chaîne. Spécifie s'il est nécessaire d'écrire un descripteur à deux octets pour la chaîne décrivant la longueur. La valeur par défaut est **False**.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	63	*RecordNumber* est inférieur à 1 et différent de -1.
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

FileGet est seulement valide en mode **Random** et **Binary**.

Les données lues à l'aide de la fonction **FileGet** sont généralement écrites dans un fichier avec la fonction **FilePut**.

Le premier enregistrement ou octet d'un fichier occupe la position 1, le deuxième enregistrement ou octet la position 2, et ainsi de suite. Si l'argument *RecordNumber* est omis, la lecture commence à partir de l'enregistrement ou de l'octet suivant la dernière fonction **FileGet** ou **FilePut** (ou à partir de celui désigné par la dernière fonction **Seek**).

Remarque de sécurité
Lors de la lecture de fichiers, ne décidez pas du contenu d'un fichier en vous basant sur l'extension de son nom. Par exemple, un fichier nommé Form1.vb peut ne pas être un fichier source Visual Basic.

Mode Random

Les règles suivantes s'appliquent aux fichiers ouverts en mode **Random** :

- Si la longueur des données lues est inférieure à la longueur spécifiée dans la clause *RecordLength* de la fonction

FileOpen, la fonction **FileGet** poursuit la lecture des enregistrements suivants conformément aux limites de longueur des enregistrements. L'espace compris entre la fin d'un enregistrement et le début de l'enregistrement suivant est rempli par le contenu de la mémoire tampon du fichier. Comme il est impossible de déterminer avec certitude le volume des données de remplissage, il est généralement préférable que la longueur des enregistrements corresponde à la longueur des données lues.

- Si la variable lue est une chaîne, la fonction **FileGet** lit par défaut un descripteur à deux octets contenant la longueur de la chaîne, puis elle lit les données qui sont placées dans la variable. La longueur d'enregistrement spécifiée par la clause *RecordLength* de la fonction **FileOpen** doit donc être supérieure d'au moins deux octets à la longueur réelle de la chaîne. Visual Basic 6.0 et les versions antérieures prennent en charge les chaînes de longueur fixe ; lors de leur écriture dans un fichier, le descripteur de longueur n'est pas écrit. Si vous souhaitez lire une chaîne sans descripteur, vous devez passer la valeur **True** au paramètre *StringIsFixedLength* de sorte que la chaîne lue soit de la longueur voulue.

- Si la variable lue est un tableau, vous pouvez au besoin lire un descripteur de la taille et des dimensions du tableau. Pour écrire le descripteur, affectez la valeur **True** au paramètre *ArrayIsDynamic*. Lors de la lecture du tableau, vous devez vous conformer à la manière dont il a été écrit. Si le tableau a été écrit avec le descripteur, vous devez lire le descripteur. Si le descripteur n'est pas employé, la taille et les limites du tableau passées à **FileGet** déterminent ce qu'il faut lire.

  Le descripteur spécifie le rang du tableau, la taille et les limites inférieures de chaque rang. Sa longueur est égale à 2 plus 8 fois le nombre de dimensions : *(2 + 8 * NumberOfDimensions)*. La longueur d'enregistrement spécifiée par le paramètre *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la somme de tous les octets nécessaires à l'écriture des données et du descripteur du tableau. Par exemple, la déclaration de tableau suivante nécessite 118 octets pour l'écriture du tableau sur le disque.

  VB

  ```
  Dim MyArray(4,9) As Integer
  ```

  Les 118 octets sont distribués comme suit :

  - 18 octets pour le descripteur : *(2 + 8 * 2)*
  - 100 octets pour les données : *(5 * 10 * 2)*.

- Si la variable lue est un autre type de variable (ni une chaîne de longueur variable ni un objet), la fonction **FileGet** ne lit que les données variables. La longueur d'enregistrement spécifiée par la clause *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la longueur des données lues.

- La fonction **FileGet** lit les éléments de structures comme si chacun d'eux était lu de façon individuelle, à ceci près qu'il n'y a pas de remplissage entre les éléments. Sur un disque, un tableau dynamique d'un type défini par l'utilisateur (écrit avec la fonction **FilePut**) est préfixé d'un descripteur dont la longueur est égale à 2 plus 8 fois le nombre de dimensions : *(2 + 8 * NumberOfDimensions)*. La longueur d'enregistrement spécifiée par la clause *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la somme de tous les octets nécessaires à la lecture des éléments individuels, incluant les tableaux et leurs descripteurs. L'attribut **VBFixedString** peut être appliqué aux champs de type chaîne dans les structures pour indiquer la taille d'une chaîne lors de l'écriture sur le disque.

Mode Binary

Pour les fichiers ouverts en mode **Binary**, la plupart des règles du mode **Random** s'appliquent, à quelques exceptions près. Les règles suivantes pour les fichiers ouverts en mode **Binary** diffèrent des règles pour les fichiers ouverts en mode **Random** :

- La clause *RecordLength* de la fonction **FileOpen** n'a aucun effet. La fonction **FileGet** lit toutes les variables du disque de manière contiguë, c'est-à-dire sans remplissage entre les enregistrements.

- Pour tout tableau autre que les tableaux d'une structure, la fonction **FileGet** ne lit que les données. Elle ne lit aucun descripteur.

- **FileGet** lit les chaînes de longueur variable qui ne sont pas des éléments de structures, sans rechercher le descripteur de longueur à deux octets. Le nombre d'octets lus est égal au nombre de caractères de la chaîne.

Remarque de sécurité
La lecture d'un fichier avec la fonction **FileGet** requiert l'accès **Read** de l'énumération FileIOPermissionAccess.

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileOpen, fonction
FilePut, fonction
Seek, fonction
FileGetObject, fonction

Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic

FileGetObject, fonction

Lit les données d'un fichier disque ouvert et les place dans une variable.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FileGetObject**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Object, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Double, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As System.Array, _
```

```
    Optional RecordNumber As Integer = -1, _
    Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Overloads Public Sub FileGetObject( _
    ByVal FileNumber As Integer, _
    ByRef Value As String, _
    Optional RecordNumber As Integer = -1, _
    Optional StringIsFixedLength As Boolean = False _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Value

Requis. Nom de variable valide destiné à recevoir les données lues.

RecordNumber

Facultatif. Numéro de l'enregistrement (fichiers ouverts en mode **Random**) ou de l'octet (fichiers ouverts en mode **Binary**) par lequel la lecture débute.

ArrayIsDynamic

Facultatif. S'applique uniquement à l'écriture d'un tableau. Spécifie si le tableau doit être considéré comme dynamique et par conséquent s'il est nécessaire d'écrire un descripteur de tableau décrivant la taille et les limites du tableau.

StringIsFixedLength

Facultatif. S'applique uniquement lors de l'écriture d'une chaîne. Spécifie s'il est nécessaire d'écrire un descripteur à deux octets pour la chaîne décrivant la longueur. La valeur par défaut est **False**.

Notes

La fonction **FileGetObject** est utilisée à la place de **FileGet** afin d'éviter toute ambiguïté au moment de la compilation si le type **Object** est retourné au lieu d'un autre type, comme **Integer**, **Long**, **Short**, etc.

Si vous avez l'intention d'écrire le type **Variant**, la fonction **FileGetObject** est requise. En cas de doute, sachez que si vous utilisez un objet pour le deuxième paramètre, il est toujours plus sûr d'utiliser **FilePutObject** et **FileGetObject**.

FileGetObject est seulement valide en mode **Random** et **Binary**.

Les données lues à l'aide de la fonction **FileGetObject** sont généralement écrites avec la fonction **FilePutObject**.

Le premier enregistrement ou octet d'un fichier occupe la position 1, le deuxième enregistrement ou octet la position 2, et ainsi de suite. Si l'argument *RecordNumber* est omis, **FileGetObject** commence à lire l'enregistrement ou l'octet qui suit la dernière fonction **FileGetObject** ou **FilePutObject** (ou l'enregistrement ou l'octet désigné par la dernière fonction **Seek**).

Mode Random

Les règles suivantes s'appliquent aux fichiers ouverts en mode **Random** :

- Si la longueur des données lues est inférieure à la longueur spécifiée dans la clause *RecordLength* de la fonction **FileOpen**, la fonction **FileGetObject** poursuit la lecture des enregistrements suivants conformément aux limites de longueur des enregistrements. L'espace compris entre la fin d'un enregistrement et le début de l'enregistrement suivant est rempli par le contenu de la mémoire tampon du fichier. Comme il est impossible de déterminer avec précision le volume des données de remplissage, il est généralement préférable que la longueur des enregistrements corresponde à la longueur des données lues.
- Si la variable lue est une chaîne, la fonction **FileGetObject** lit par défaut un descripteur à deux octets contenant la longueur de la chaîne, puis elle lit les données qui sont placées dans la variable. La longueur d'enregistrement spécifiée par la clause *RecordLength* de la fonction **FileOpen** doit donc être supérieure d'au moins deux octets à la longueur réelle de la chaîne. Visual Basic 6.0 et les versions antérieures prennent en charge les chaînes de longueur fixe et, lors de leur lecture dans un fichier, le descripteur de longueur n'est pas écrit. Si vous souhaitez lire une chaîne sans descripteur, vous devez passer la valeur **True** au paramètre *StringIsFixedLength* de sorte que la chaîne lue soit de la longueur voulue.
- Si la variable lue est un tableau, vous pouvez au besoin lire un descripteur de la taille et des dimensions du tableau. Pour

lire le descripteur, affectez la valeur *ArrayIsDynamic* au paramètre **True**. Lors de la lecture du tableau, vous devez vous conformer à la manière dont il a été écrit. Si le tableau a été écrit avec le descripteur, vous devez lire le descripteur. Si le descripteur n'est pas employé, la taille et les limites du tableau passées à **FileGetObject** sont utilisées pour déterminer ce qu'il faut lire.

Le descripteur spécifie le rang du tableau, la taille et les limites inférieures de chaque rang. Sa longueur est égale à 2 plus 8 fois le nombre de dimensions : *2 + 8 * NumberOfDimensions*. La longueur d'enregistrement spécifiée par le paramètre *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la somme de tous les octets nécessaires à l'écriture des données et du descripteur du tableau. Par exemple, la déclaration de tableau suivante nécessite 118 octets pour l'écriture du tableau sur le disque :

VB

```
Dim MyArray(4,9) As Integer
```

Les 118 octets se décomposent de la façon suivante : 18 octets pour le descripteur (2 + 8 * 2) et 100 octets pour les données (5 * 10 * 2).

- La fonction **FileGetObject** lit les éléments de structures comme si chacun d'eux était lu de façon individuelle, à ceci près qu'il n'y a pas de remplissage entre les éléments. Sur un disque, un tableau dynamique d'un type défini par l'utilisateur (écrit avec la fonction **FilePutObject**) est préfixé d'un descripteur dont la longueur est égale à 2 plus 8 fois le nombre de dimensions : *2 + 8 * NumberOfDimensions*. La longueur d'enregistrement spécifiée par la clause *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la somme de tous les octets nécessaires à la lecture des éléments individuels, incluant les tableaux et leurs descripteurs. La classe VBFixedStringAttribute peut être appliquée aux champs de type chaîne dans les structures pour indiquer la taille de la chaîne lors de l'écriture sur le disque.

Mode Binary

Pour les fichiers ouverts en mode **Binary**, toutes les règles **Random** s'appliquent, à l'exception de ce qui suit :

- La clause *RecordLength* de la fonction **FileOpen** n'a aucun effet. La fonction **FileGetObject** lit toutes les variables du disque de manière contiguë, c'est-à-dire sans remplissage entre les enregistrements.
- Pour tout tableau autre que les tableaux d'une structure, la fonction **FileGetObject** ne lit que les données. Elle ne lit aucun descripteur.

FileGetObject lit les chaînes de longueur variable qui ne sont pas des éléments de structures, sans rechercher le descripteur de longueur à deux octets. Le nombre d'octets lus est égal au nombre de caractères de la chaîne.

Remarque de sécurité
Lors de la lecture de fichiers, ne décidez pas du contenu d'un fichier en vous basant sur l'extension de son nom. Par exemple, un fichier nommé Form1.vb peut ne pas être un fichier source Visual Basic.

Exemple

L'exemple suivant lit un enregistrement dans un fichier de test, puis le récupère.

VB

```
Dim c As Object = "test"
FileSystem.FileOpen(1, "test.dat", OpenMode.Binary)
FileSystem.FilePutObject(1, "ABCDEF")
FileSystem.Seek(1, 1)
FileSystem.FileGetObject(1, c)
MsgBox(c)
FileSystem.FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FilePut, fonction

FileOpen, fonction

Seek, fonction

FileGet, fonction

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

FileLen, fonction

Retourne une valeur **Long** spécifiant la longueur d'un fichier en octets.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FileLen**. Pour plus d'informations, consultez My.Computer.FileSystem.GetFileInfo, méthode.

```
Public Function FileLen(ByVal PathName As String) As Long
```

Paramètres
PathName

Requis. Expression de type **String** qui spécifie un fichier. *PathName* peut comprendre le répertoire ou le dossier, et le lecteur.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
FileNotFoundException	53	Le fichier n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si le fichier spécifié est ouvert lorsque la fonction **FileLen** est appelée, la valeur retournée représente la taille du fichier au moment de son ouverture.

Remarque

Pour obtenir la longueur actuelle d'un fichier ouvert, utilisez la fonction **LOF**.

Exemple

Cet exemple utilise la fonction **FileLen** pour retourner la longueur d'un fichier en octets. Pour cet exemple, supposons que `TestFile` est un fichier contenant des données.

VB

```
Dim MySize As Long
' Returns file length (bytes).
MySize = FileLen("TESTFILE")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence
FileDateTime, fonction
GetAttr, fonction
LOF, fonction
FileNotFoundException Class

Autres ressources
Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

FileOpen, fonction

Ouvre un fichier pour des opérations d'entrée ou de sortie.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FileOpen**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub FileOpen( _
   ByVal FileNumber As Integer, _
   ByVal FileName As String, _
   ByVal Mode As OpenMode, _
   Optional ByVal Access As OpenAccess = OpenAccess.Default, _
   Optional ByVal Share As OpenShare = OpenShare.Default, _
   Optional ByVal RecordLength As Integer = -1 _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide. Utilisez la fonction **FreeFile** pour obtenir le prochain numéro de fichier disponible.

FileName

Requis. Expression **String** qui spécifie un nom de fichier, peut comprendre un répertoire ou un dossier, et un lecteur.

Mode

Requis. Énumération spécifiant le mode d'accès au fichier : **Append**, **Binary**, **Input**, **Output** ou **Random**. (Pour plus d'informations, consultez OpenMode, énumération).

Access

Facultatif. Énumération spécifiant les opérations autorisées sur le fichier ouvert : **Read**, **Write** ou **ReadWrite**. Valeurs par défaut affectées à **ReadWrite**. (Pour plus d'informations, consultez OpenAccess, énumération).

Share

Facultatif. Énumération spécifiant les opérations non autorisées sur le fichier ouvert par d'autres processus : **Shared**, **Lock Read**, **Lock Write** et **Lock Read Write**. Valeurs par défaut affectées à **Lock Read Write**. (Pour plus d'informations, consultez OpenShare, énumération).

RecordLength

Facultatif. Nombre inférieur ou égal à 32 767 (octets). Pour les fichiers ouverts en mode Random, cette valeur représente la longueur de l'enregistrement. Pour les fichiers séquentiels, elle représente le nombre de caractères contenus dans la mémoire tampon.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	**Access**, **Share** ou **Mode** non valide.
ArgumentException	5	Le fichier **WriteOnly** est ouvert pour **Input**.
ArgumentException	5	Le fichier **ReadOnly** est ouvert pour **Output**.
ArgumentException	5	Le fichier **ReadOnly** est ouvert pour **Append**.

ArgumentException	5	La longueur d'enregistrement est négative (et différente de -1).
IOException	52	*FileNumber* n'est pas valide (<-1 ou >255) ou *FileNumber* est déjà en cours d'utilisation.
IOException	55	*FileName* est déjà ouvert ou *FileName* n'est pas valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **FileOpen** est fournie pour la compatibilité descendante et peut affecter la performance. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Vous devez ouvrir un fichier avant qu'une opération d'E/S puisse être exécutée dessus. **FileOpen** alloue une mémoire tampon pour l'E/S du fichier et détermine le mode d'accès à utiliser avec la mémoire tampon.

Remarque de sécurité
Lors de l'écriture dans un fichier, une application devra peut-être créer un fichier, si le fichier dans lequel elle essaie d'écrire n'existe pas. Pour ce faire, elle nécessite une autorisation sur le répertoire dans lequel le fichier doit être créé. Cependant, si le fichier spécifié par *FileName* existe, l'application n'a besoin que de l'autorisation **Write** sur ce fichier. Dans la mesure du possible, et pour plus de sécurité, créez le fichier au moment du déploiement et accordez une autorisation **Write** pour ce fichier uniquement plutôt que pour l'ensemble du répertoire. Pour plus de sécurité, écrivez des données dans des répertoires utilisateur plutôt que dans le répertoire racine ou le répertoire Program Files.

Le canal à ouvrir peut être trouvé à l'aide de la fonction **FreeFile()**.

Remarque de sécurité
La fonction **FileOpen** nécessite un accès **Read** de l'énumération **FileIOPermissionAccess**, ce qui peut affecter son exécution dans les situations où le niveau de confiance n'est pas total. Pour plus d'informations, consultez FileIOPermissionAccess et Demande d'autorisations.

Exemple

Cet exemple illustre différentes utilisations possibles de la fonction **FileOpen** pour permettre l'entrée et la sortie dans un fichier.

Le code suivant ouvre le fichier `TestFile` en mode **Input**.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Input)
' Close before reopening in another mode.
FileClose(1)
```

Cet exemple ouvre le fichier en mode **Binary** pour des opérations d'écriture uniquement.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Binary,OpenAccess.Write)
' Close before reopening in another mode.
FileClose(1)
```

L'exemple suivant ouvre le fichier en mode **Random**. Le fichier contient les enregistrements de la structure `Person`.

VB

```
Structure Person
    <VBFixedString(30)> Dim Name As String
    Dim ID As Integer
End Structure
Public Sub ExampleMethod()
    ' Count 30 for the string, plus 4 for the integer.
    FileOpen(1, "TESTFILE", OpenMode.Random, , , 34)
    ' Close before reopening in another mode.
    FileClose(1)
End Sub
```

Cet exemple de code ouvre le fichier en mode **Output** ; n'importe quel processus peut lire ou écrire dans le fichier.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output, OpenAccess.Default, OpenShare.Shared)
' Close before reopening in another mode.
FileClose(1)
```

Cet exemple de code ouvre le fichier en mode **Binary** pour la lecture ; les autres processus ne peuvent pas le lire.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Binary, OpenAccess.Read, _
   OpenShare.LockRead)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileClose, fonction

FreeFile, fonction

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

FilePut, fonction

Écrit les données d'une variable dans un fichier disque.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FilePut**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Double, _
      RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As System.Array, _
   Optional RecordNumber As Integer = -1, _
   Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Public Overloads Sub FilePut (
      FileNumber As Integer,
```

```
        Value As String, _
    Optional RecordNumber As Integer = -1,
    Optional StringIsFixedLength As Boolean = False
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Value

Requis. Nom de variable valide contenant les données écrites sur le disque.

RecordNumber

Facultatif. Numéro de l'enregistrement (fichiers ouverts en mode **Random**) ou de l'octet (fichiers ouverts en mode **Binary**) qui marque le début de l'écriture.

ArrayIsDynamic

Facultatif. S'applique uniquement à l'écriture d'un tableau. Spécifie si le tableau doit être considéré comme dynamique et s'il est nécessaire d'écrire un descripteur de tableau pour la chaîne décrivant la longueur.

StringIsFixedLength

Facultatif. S'applique uniquement lors de l'écriture d'une chaîne. Spécifie s'il est nécessaire d'écrire dans le fichier un descripteur à deux octets pour la chaîne décrivant la longueur. La valeur par défaut est **False**.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	63	*RecordNumber* est inférieur à 1 et différent de -1.
IOExceptio	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

FilePut est seulement valide en mode **Random** et **Binary**.

Les données écrites avec **FilePut** sont généralement lues à partir d'un fichier à l'aide de **FileGet**.

Le premier enregistrement ou octet d'un fichier occupe la position 1, le deuxième enregistrement ou octet la position 2, et ainsi de suite. Si l'argument *RecordNumber* est omis, l'écriture commence à partir de l'enregistrement ou de l'octet suivant la dernière fonction **FileGet** ou **FilePut** (ou à partir de celui désigné par la dernière fonction **Seek**).

L'argument *StringIsFixedLength* contrôle si la fonction interprète des chaînes comme variable ou à longueur fixe. **FilePut** n'écrit pas le descripteur de longueur lorsque l'argument est **True**. Si vous utilisez *StringIsFixedLength* = **True** avec **FilePut**, vous devez en faire de même avec **FileGet** et vous devez en outre vous assurer que la chaîne est initialisée à la longueur attendue.

Mode Random

Les règles suivantes s'appliquent aux fichiers ouverts en mode **Random** :

- Si la longueur des données écrites est inférieure à la longueur spécifiée dans la clause *RecordLength* de la fonction **FileOpen**, la fonction **FilePut** poursuit l'écriture des enregistrements suivants conformément aux limites de longueur des enregistrements. L'espace compris entre la fin d'un enregistrement et le début de l'enregistrement suivant est rempli par le contenu de la mémoire tampon du fichier. Comme il est impossible de déterminer avec certitude le volume des données de remplissage, il est généralement préférable que la longueur des enregistrements corresponde à la longueur des données écrites. Une exception est levée si la longueur des données écrites est supérieure à celle spécifiée dans la clause *RecordLength* de la fonction **FileOpen**.

- Si la variable écrite est une chaîne, la fonction **FilePut** écrit d'abord un descripteur à deux octets contenant la longueur de la chaîne, puis les données qui sont placées dans la variable. La longueur d'enregistrement spécifiée par la clause *RecordLength* de la fonction **FileOpen** doit donc être supérieure d'au moins deux octets à la longueur réelle de la chaîne.
- Si la variable écrite est un objet contenant un type numérique, la fonction **FilePut** écrit d'abord deux octets qui identifient le **VarType** de l'objet puis la variable. Par exemple, lorsque vous écrivez un objet contenant un entier, **FilePut** écrit six octets : deux octets identifiant l'objet comme **VarType(3)** (**Integer**) et quatre octets contenant les données. La longueur d'enregistrement spécifiée par le paramètre *RecordLength* dans la fonction **FileOpen** doit être supérieure d'au moins deux octets au nombre réel d'octets requis pour stocker la variable.
- Si la variable écrite est un objet contenant une chaîne, **FilePut** écrit un descripteur à deux octets identifiant le **VarType(8)** de l'objet, un descripteur à deux octets indiquant la longueur de la chaîne, puis la chaîne de données. La longueur d'enregistrement spécifiée par le paramètre *RecordLength* dans la fonction **FileOpen** doit être supérieure d'au moins quatre octets à la longueur réelle de la chaîne. Si vous souhaitez placer une chaîne sans descripteur, vous devez passer la valeur **True** au paramètre *StringIsFixedLength* de sorte que la chaîne lue soit de la longueur voulue.
- Si la variable écrite est un tableau, vous avez la possibilité d'écrire ou non un descripteur pour la taille et les dimensions du tableau. Visual Basic 6.0 et les versions antérieures écrivent le descripteur de fichier pour un tableau dynamique, mais pas pour un tableau de taille fixe. Par défaut, Visual Basic 2005 n'écrit pas le descripteur. Pour écrire le descripteur, affectez la valeur **True** au paramètre *ArrayIsDynamic*. Lorsque vous écrivez le tableau, vous devez faire correspondre le mode de lecture du tableau ; s'il est lu par le descripteur, vous devez écrire ce dernier. Le descripteur spécifie le rang du tableau, la taille et les limites inférieures de chaque rang. Sa longueur est égale à 2 plus 8 fois le nombre de dimensions : *(2 + 8 * NumberOfDimensions)*. La longueur d'enregistrement spécifiée par la clause *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la somme de tous les octets nécessaires à l'écriture des données et du descripteur du tableau. Par exemple, la déclaration de tableau suivante nécessite 118 octets pour l'écriture du tableau sur le disque.

  VB
  ```
  Dim MyArray(4,9) As Integer
  ```
- Si la variable écrite est d'un type différent (ni une chaîne de longueur variable ni un objet), la fonction **FilePut** n'écrit que les données variables. La longueur d'enregistrement spécifiée par la clause *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la longueur des données écrites.
- **FilePut** écrit les éléments de structures comme si chacun d'entre eux était écrit individuellement, sauf qu'il n'y a pas de remplissage entre les éléments. L'attribut **VBFixedString** peut être appliqué aux champs de type chaîne dans les structures pour indiquer la taille de la chaîne lors de l'écriture sur le disque.

  Remarque

  Les champs de chaîne dont le nombre d'octets est supérieur au nombre spécifié par l'attribut **VBFixedString** sont tron qués lors de l'écriture sur le disque,

Mode Binary

Pour les fichiers ouverts en mode **Binary**, la plupart des règles du mode **Random** s'appliquent, à quelques exceptions près. Les règles suivantes pour les fichiers ouverts en mode **Binary** diffèrent des règles appliquées au mode **Random** :

- La clause *RecordLength* de la fonction **FileOpen** n'a aucun effet. La fonction **FilePut** écrit toutes les variables sur le disque de manière contiguë, c'est-à-dire sans remplissage entre les enregistrements.
- Pour tout tableau autre que ceux d'une structure, la fonction **FilePut** n'écrit que les données. Elle n'écrit aucun descripteur.
- La fonction **FilePut** écrit les chaînes de longueur variable qui ne sont pas des éléments de structure, sans le descripteur à deux octets. Le nombre d'octets écrits est égal au nombre de caractères de la chaîne. Par exemple, les instructions suivantes écrivent 11 octets dans le fichier numéro 1 :

  VB
  ```
  Dim hellow As String = "Hello World"
  FilePut(1,hellow)
  ```

- L'écriture sur fichier à l'aide de la fonction **FilePut** requiert l'accès **Write** à partir de l'énumération FileIOPermissionAccess.

Exemple

Cet exemple utilise la fonction **FilePut** pour écrire des données dans un fichier. Cinq enregistrements de la structure `Person` sont écrits dans le fichier.

VB

```
Structure Person
   Public ID As Integer
   Public Name As String
End Structure

Sub WriteData()
   Dim PatientRecord As Person
   Dim recordNumber As Integer
'    Open file for random access.
   FileOpen(1, "C:\TESTFILE.txt", OpenMode.Binary)
   ' Loop 5 times.
   For recordNumber = 1 To 5
      ' Define ID.
      PatientRecord.ID = recordNumber
      ' Create a string.
      PatientRecord.Name = "Name " & recordNumber
      ' Write record to file.
      FilePut(1, PatientRecord)
   Next recordNumber
   FileClose(1)
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileGet, fonction
FileOpen, fonction
Seek, fonction
FileGetObject, fonction
VBFixedStringAttribute, classe
Len, fonction (Visual Basic)
ArgumentException
IOException

Autres ressources

Écriture dans des fichiers en Visual Basic

FilePutObject, fonction

Écrit les données d'une variable dans un fichier disque.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FilePutObject**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Object, _
      RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Double, _
      RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As System.Array, _
```

```
    Optional RecordNumber As Integer = -1, _
    Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As String, _
    Optional RecordNumber As Integer = -1, _
    Optional StringIsFixedLength As Boolean = False _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Value

Requis. Nom de variable valide contenant les données écrites sur le disque.

RecordNumber

Facultatif. Numéro de l'enregistrement (fichiers ouverts en mode **Random**) ou de l'octet (fichiers ouverts en mode **Binary**) par lequel l'écriture débute.

ArrayIsDynamic

Facultatif. S'applique uniquement à l'écriture d'un tableau. Spécifie si le tableau doit être considéré comme dynamique et s'il est nécessaire d'écrire un descripteur de tableau pour la chaîne décrivant la longueur.

StringIsFixedLength

Facultatif. S'applique uniquement lors de l'écriture d'une chaîne. Spécifie s'il est nécessaire d'écrire un descripteur pour la chaîne indiquant la longueur. La valeur par défaut est **False**.

Notes

La fonction **FilePutObject** est utilisée à la place de **FilePut** afin d'éviter toute ambiguïté au moment de la compilation si le type **Object** est passé au lieu d'un autre type, tel que **Integer**, **Long**, **Short**, etc.

FilePutObject écrit et lit des descripteurs décrivant l'objet. Si vous avez l'intention d'écrire le type **Variant**, **FilePutObject** est requis. En cas de doute, sachez que si vous utilisez un objet pour le deuxième paramètre, il est toujours plus sûr d'utiliser **FilePutObject** et **FileGetObject**.

FilePutObject est seulement valide en mode **Random** et **Binary**.

Les données écrites avec **FilePutObject** sont généralement lues à partir d'un fichier avec la fonction **FileGetObject**.

Le premier enregistrement ou octet d'un fichier occupe la position 1, le deuxième enregistrement ou octet la position 2, et ainsi de suite. Si l'argument *RecordNumber* est omis, **FilePutObject** écrit l'enregistrement ou l'octet à la suite de la dernière fonction **FileGetObject** ou **FilePutObject** (ou l'enregistrement ou l'octet désigné par la dernière fonction **Seek**).

L'argument *StringIsFixedLength* contrôle si la fonction interprète des chaînes comme variable ou à longueur fixe. **FilePutObject** n'écrit pas le descripteur de longueur lorsque l'argument est **True**. Si vous employez *StringIsFixedLength* = **True** avec **FilePutObject**, vous devez en faire de même avec **FileGetObject** et vous devez en outre vous assurer que la chaîne est initialisée à la longueur attendue.

Mode Random

Les règles suivantes s'appliquent aux fichiers ouverts en mode **Random** :

- Si la longueur des données écrites est inférieure à la longueur spécifiée dans la clause *RecordLength* de la fonction **FileOpen**, la fonction **FilePutObject** poursuit l'écriture des enregistrements suivants conformément aux limites de longueur des enregistrements. L'espace compris entre la fin d'un enregistrement et le début de l'enregistrement suivant est rempli par le contenu de la mémoire tampon du fichier. Comme il est impossible de déterminer avec précision le volume des données de remplissage, il est généralement préférable que la longueur des enregistrements corresponde à la longueur des données écrites. Une exception est levée si la longueur des données écrites est supérieure à celle spécifiée dans la clause *RecordLength* de la fonction **FileOpen**.
- Si la variable écrite est un objet contenant un type numérique, **FilePutObject** écrit d'abord deux octets qui identifient le **VarType** de l'objet puis la variable. Par exemple, lorsque vous écrivez un objet contenant un entier, **FilePutObject** écrit

six octets : deux octets identifiant l'objet comme **VarType(**3**)** (**Integer**) et quatre octets contenant les données. La longueur d'enregistrement spécifiée par le paramètre *RecordLength* dans la fonction **FileOpen** doit être supérieure d'au moins deux octets au nombre réel d'octets requis pour stocker la variable.

- Si la variable écrite est un objet contenant une chaîne, **FilePutObject** écrit un descripteur à deux octets identifiant le **VarType(**8**)** de l'objet, un descripteur à deux octets indiquant la longueur de la chaîne, puis la chaîne de données. La longueur d'enregistrement spécifiée par le paramètre *RecordLength* dans la fonction **FileOpen** doit être supérieure d'au moins quatre octets à la longueur réelle de la chaîne. Si vous souhaitez placer une chaîne sans descripteur, vous devez passer la valeur **True** au paramètre *StringIsFixedLength* de sorte que la chaîne lue soit de la longueur voulue.
- Si la variable écrite est un tableau, vous pouvez au besoin écrire un descripteur de la taille et des dimensions du tableau. Visual Basic 6 et les versions antérieures auraient écrit le descripteur de fichier dans le cas d'un tableau dynamique, mais pas pour un tableau de taille fixe. Par défaut, Visual Basic n'écrit pas ce descripteur. Pour écrire le descripteur, affectez la valeur **True** au paramètre *ArrayIsDynamic*. Le mode d'écriture du tableau doit absolument concorder avec son mode de lecture : si le tableau est lu avec le descripteur, vous devez écrire le descripteur. Le descripteur spécifie le rang du tableau, la taille et les limites inférieures de chaque rang. Sa longueur est égale à 2 plus 8 fois le nombre de dimensions : *(2 + 8 * NumberOfDimensions)*. La longueur d'enregistrement spécifiée par la clause *RecordLength* dans la fonction **FileOpen** doit être supérieure ou égale à la somme de tous les octets nécessaires à l'écriture des données et du descripteur du tableau.

Mode Binary

Pour les fichiers ouverts en mode **Binary**, toutes les règles du mode **Random** s'appliquent, à l'exception de ce qui suit :

- La clause *RecordLength* de la fonction **FileOpen** n'a aucun effet. La fonction **FilePutObject** écrit toutes les variables sur le disque de manière contiguë, c'est-à-dire sans remplissage entre les enregistrements.

Exemple

Cet exemple utilise la fonction **FilePutObject** pour écrire une chaîne dans un fichier.

VB

```
Sub WriteData()
    Dim text As String = "test"
    FileOpen(1, "test.bin", OpenMode.Binary)
    FilePutObject(1, text)
    FileClose(1)
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileGet, fonction
FileOpen, fonction
Seek, fonction
FilePut, fonction
My.Computer.FileSystem, objet

Autres ressources

Écriture dans des fichiers en Visual Basic

FileWidth, fonction

Assigne une largeur de ligne de sortie à un fichier ouvert à l'aide de la fonction **FileOpen**.

```
Public Sub FileWidth( _
      FileNumber As Integer, _
      RecordWidth As Integer _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

RecordWidth

Requis. Expression numérique comprise entre 0 et 255, inclus, qui indique le nombre de caractères qui apparaissent sur une ligne avant le passage à la ligne suivante. Si la valeur de *RecordWidth* est 0, la longueur de la ligne est illimitée. La valeur par défaut *RecordWidth* est 0.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Cette fonction assigne une largeur de ligne de sortie à un fichier ouvert à l'aide de la fonction **FileOpen**.

Exemple

Cet exemple utilise la fonction **FileWidth** pour définir la longueur de la ligne de sortie d'un fichier.

VB

```
Dim i As Integer
FileOpen(1, "TESTFILE", OpenMode.Output) ' Open file for output.
FileWidth(1, 5)   ' Set output line width to 5.
For i = 0 To 9   ' Loop 10 times.
   Print(1, Chr(48 + I))   ' Prints five characters per line.
Next
FileClose(1)   ' Close file.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileOpen, fonction
Fonctions Print, PrintLine
IOException Class

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

Filter, fonction (Visual Basic)

Retourne un tableau de base zéro et contenant un sous-ensemble d'un tableau de chaînes (**String**) basé sur des critères de filtre spécifiés.

```
Function Filter(
   ByVal Source() As { Object | String },
   ByVal Match As String,
   Optional ByVal Include As Boolean = True,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
)  As String()
```

Paramètres

Source

Requis. Tableau de chaînes à une dimension dans lequel la recherche doit être effectuée.

Match

Requis. Chaîne à rechercher.

Include

Facultatif. Valeur de type **Boolean** indiquant s'il faut retourner des sous-chaînes incluant ou excluant l'argument *Match*. Si l'argument *Include* a la valeur **True**, la fonction **Filter** retourne le sous-ensemble du tableau contenant l'argument *Match* comme sous-chaîne. Si l'argument *Include* a la valeur **False**, la fonction **Filter** retourne le sous-ensemble du tableau qui ne contient pas l'argument *Match* comme sous-chaîne.

Compare

Facultatif. Valeur numérique indiquant le type de comparaison de chaîne à utiliser. Consultez la section « Paramètres » pour obtenir les valeurs.

Paramètres

L'argument *Compare* peut prendre les valeurs suivantes.

Constante	Description
CompareMethod.Binary	Effectue une comparaison binaire.
CompareMethod.Text	Effectue une comparaison de texte.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	9	*Source* a la valeur **Nothing** ou n'est pas un tableau unidimensionnel.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si aucune correspondance de *Match* n'est obtenue dans *Source*, la fonction **Filter** retourne un tableau vide. Une erreur se produit si l'argument *Source* est **Nothing** ou s'il ne s'agit pas d'un tableau à une dimension.

Le tableau retourné par la fonction **Filter** comporte uniquement le nombre d'éléments suffisants pour contenir le nombre d'éléments de travail correspondants.

Exemple

Cet exemple illustre l'utilisation de la fonction **Filter**.

VB

```
Dim TestStrings(2) As String
TestStrings(0) = "This"
TestStrings(1) = "Is"
TestStrings(2) = "It"
Dim subStrings() As String
' Returns ["This", "Is"].
subStrings = Filter(TestStrings, "is", True, CompareMethod.Text)
' Returns ["This"].
subStrings = Filter(TestStrings, "is", True, CompareMethod.Binary)
' Returns ["Is", "It"].
subStrings = Filter(TestStrings, "is", False, CompareMethod.Binary)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Replace, fonction (Visual Basic)
ArgumentException

Fonctions Int, Fix (Visual Basic)

Retournent la partie entière d'un nombre.

```
Public Shared Function Int( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
Public Shared Function Fix( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
```

Paramètres

Number

Requis. Nombre de type **Double** ou toute expression numérique valide. Si l'argument *Number* contient une valeur de type **Nothing**, **Nothing** est retourné.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	5	Number n'est pas spécifié.
ArgumentException	5	Number n'est pas un type numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les fonctions **Int** et **Fix** suppriment la partie fractionnaire de *Number* et retournent la valeur entière résultante.

La différence entre les fonctions **Int** et **Fix** réside dans le fait que si *Number* est négatif, **Int** retourne le premier entier négatif inférieur ou égal à *Number*, alors que **Fix** retourne le premier entier négatif supérieur ou égal à *Number*. Par exemple, **Int** convertit -8,4 en -9 et **Fix** convertit -8,4 en -8.

`Fix(number)` équivaut à `Sign(number) * Int(Abs(number))`.

Exemple

Cet exemple illustre de quelle façon les fonctions **Int** et **Fix** retournent la partie entière des nombres. Si l'argument est négatif, la fonction **Int** retourne le premier entier négatif inférieur ou égal au nombre ; la fonction **Fix** retourne le premier entier négatif supérieur ou égal au nombre. L'exemple suivant nécessite que vous spécifiiez **Option Strict Off**, car les conversions implicites du type **Double** en type **Integer** ne sont pas autorisées sous **Option Strict On** :

VB

```
' This code requires Option Strict Off
Dim MyNumber As Integer
MyNumber = Int(99.8)   ' Returns 99.
MyNumber = Fix(99.8)   ' Returns 99.

MyNumber = Int(-99.8)  ' Returns -100.
MyNumber = Fix(-99.8)  ' Returns -99.

MyNumber = Int(-99.2)  ' Returns -100.
MyNumber = Fix(-99.2)  ' Returns -99.
```

Vous pouvez utiliser la fonction **CInt** pour convertir explicitement d'autres types de données en type **Integer** avec **Option Strict Off**. Toutefois, **CInt** est arrondi à l'entier le plus proche au lieu de tronquer la partie fractionnaire des nombres. Par exemple :

VB

```
MyNumber = CInt(99.8)    ' Returns 100.
MyNumber = CInt(-99.8)   ' Returns -100.
MyNumber = CInt(-99.2)   ' Returns -99.
```

Vous pouvez utiliser la fonction **CInt** sur le résultat d'un appel à **Fix** ou à **Int** afin d'effectuer une conversion explicite en entier sans arrondi. Par exemple :

VB

```
MyNumber = CInt(Fix(99.8))   ' Returns 99.
MyNumber = CInt(Int(99.8))   ' Returns 99.
```

Pour plus d'informations sur **CInt**, consultez Fonctions de conversion de types de données.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Fonctions de conversion de types de données
Integer, type de données (Visual Basic)
Liste des mots clés mathématiques
Fonctions mathématiques (Visual Basic)
Liste des conversions
ArgumentNullException

Format, fonction

Retourne une chaîne mise en forme conformément aux instructions contenues dans une expression **String** de format.

```
Public Shared Function Format( _
   ByVal Expression As Object, _
   Optional ByVal Style As String = "" _
) As String
```

Paramètres

Expression

Requis. Toute expression valide.

Style

Facultatif. Expression **String** de format nommée valide ou définie par l'utilisateur.

Paramètres

Pour obtenir des informations sur la création de l'argument *Style*, consultez la rubrique appropriée répertoriée ci-dessous :

Pour mettre en forme	**Procédez comme suit**
Nombres	Utilisez des formats numériques prédéfinis ou créez des formats numériques définis par l'utilisateur.
Dates et heures	Utilisez des formats de date/heure prédéfinis ou créez des formats de date/heure définis par l'utilisateur.
Numéros de série de date et d'heure	Utilisez des formats de date et d'heure ou des formats numériques.

Si vous essayez de mettre en forme un nombre sans spécifier *Style*, la fonction **Format** fournit une fonctionnalité similaire à la fonction **Str**, bien qu'elle respecte les conventions internationales. Toutefois, les nombres positifs mis en forme en tant que chaînes à l'aide de la fonction **Format** ne contiennent pas d'espace à gauche réservé au signe de la valeur, contrairement à ceux convertis à l'aide de la fonction **Str**.

Notes

Si vous mettez en forme une chaîne numérique non localisée, vous devez utiliser un format numérique défini par l'utilisateur afin d'obtenir l'aspect voulu.

La méthode System.String.Format fournit également une fonctionnalité similaire.

Exemple

L'exemple suivant illustre différentes utilisations de la fonction **Format** pour mettre en forme des valeurs utilisant à la fois les formats **String** et les formats définis par l'utilisateur. Pour le séparateur de date (**/**), le séparateur d'heure (**:**) et les indicateurs AM/PM (**t** et **tt**), le résultat réel mis en forme affiché par votre système dépend des paramètres régionaux utilisés par le code. Lorsque les heures et les dates sont affichées dans l'environnement de développement, les formats d'heure abrégée et de date courte des paramètres régionaux de code sont utilisés.

Remarque

Pour paramètres régionaux configurés avec une horloge au format 24 heures, les indicateurs AM/PM (**t** et **tt**) n'affichent rien.

VB

```
Dim TestDateTime As Date = #1/27/2001 5:04:23 PM#
Dim TestStr As String
' Returns current system time in the system-defined long time format.
TestStr = Format(Now(), "Long Time")
' Returns current system date in the system-defined long date format.
```

```
TestStr = Format(Now(), "Long Date")
' Also returns current system date in the system-defined long date
' format, using the single letter code for the format.
TestStr = Format(Now(), "D")

' Returns the value of TestDateTime in user-defined date/time formats.
' Returns "5:4:23".
TestStr = Format(TestDateTime, "h:m:s")
' Returns "05:04:23 PM".
TestStr = Format(TestDateTime, "hh:mm:ss tt")
' Returns "Saturday, Jan 27 2001".
TestStr = Format(TestDateTime, "dddd, MMM d yyyy")
' Returns "17:04:23".
TestStr = Format(TestDateTime, "HH:mm:ss")
' Returns "23".
TestStr = Format(23)

' User-defined numeric formats.
' Returns "5,459.40".
TestStr = Format(5459.4, "##,##0.00")
' Returns "334.90".
TestStr = Format(334.9, "###0.00")
' Returns "500.00%".
TestStr = Format(5, "0.00%")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Formats différents pour valeurs numériques différentes (fonction Format)
Formats de date/heure prédéfinis (fonction Format)
Formats numériques prédéfinis (fonction Format)
Str, fonction
Fonctions de conversion de types de données
Formats de date/heure définis par l'utilisateur (fonction Format)
Formats numériques définis par l'utilisateur (fonction Format)
Format

Formats différents pour valeurs numériques différentes (fonction Format)

Une expression de format définie par l'utilisateur pour des nombres peut avoir de une à trois sections séparées par des points-virgules. Si l'argument *Style* de la fonction **Format** contient un des formats numériques prédéfinis, une seule section est autorisée.

Si vous utilisez :	**Le résultat est le suivant :**
Une seule section	L'expression de format s'applique à toutes les valeurs.
Deux sections	La première section s'applique aux valeurs positives et aux zéros ; la deuxième s'applique aux valeurs négatives.
Trois sections	La première section s'applique aux valeurs positives, la deuxième s'applique aux valeurs négatives et la troisième s'applique aux zéros.

L'exemple suivant comporte deux sections : la première définit le format pour les valeurs positives et les zéros, la deuxième définit le format pour les valeurs négatives. Compte tenu que l'argument *Style* de la fonction **Format** prend pour valeur une chaîne, il est placé entre guillemets.

VB

```
Dim Style1 As String = "$#,##0;($#,##0)"
```

Si vous incluez des points-virgules sans rien entre eux, la section manquante est imprimée à l'aide du format de la valeur positive. Par exemple, le format suivant affiche des valeurs positives et négatives en utilisant le format de la première section et affiche `Zero` si la valeur est égale à zéro.

VB

```
Dim Style2 As String = "$#,##0;;\Z\e\r\o"
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Formats numériques prédéfinis (fonction Format)

Format, fonction

Liste des manipulations de chaînes

Formats numériques prédéfinis (fonction Format)

Le tableau suivant identifie les noms de formats numériques prédéfinis. Ceux-ci peuvent être utilisés par nom comme argument de style pour la fonction **Format** :

Nom de format	Description
General Number, **G** ou **g**	Affiche le nombre sans séparateur de milliers.
Currency, **C** ou **c**	Affiche le nombre avec séparateur de milliers, le cas échéant ; affiche deux chiffres à droite du séparateur décimal. Le résultat est fonction des paramètres régionaux du système.
Fixed, **F** ou **f**	Affiche au moins un chiffre à gauche et deux chiffres à droite du séparateur décimal.
Standard, **N** ou **n**	Affiche un nombre avec séparateur de milliers, au moins un chiffre à gauche et deux chiffres à droite du séparateur décimal.
Percent	Affiche le nombre multiplié par 100 avec un signe de pourcentage (%) ajouté immédiatement à droite ; affiche toujours deux chiffres à droite du séparateur décimal.
P ou **p**	Affiche un nombre avec un séparateur de milliers multiplié par 100 avec un signe de pourcentage (%) ajouté à droite et séparé par un seul espace ; affiche toujours deux chiffres à droite du séparateur décimal.
Scientific	Utilise la notation scientifique standard, en fournissant deux chiffres significatifs.
E ou **e**	Utilise la notation scientifique standard, en fournissant six chiffres significatifs.
D ou **d**	Affiche le nombre sous forme de chaîne contenant la valeur du nombre au format décimal (base 10). Cette option n'est prise en charge que pour les types entiers (**Byte**, **Short**, **Integer**, **Long**).
X ou **x**	Affiche le nombre sous forme de chaîne contenant la valeur du nombre au format hexadécimal (base 16). Cette option n'est prise en charge que pour les types entiers (**Byte**, **Short**, **Integer**, **Long**).
Yes/No	Affiche `No` si le nombre est 0 ; sinon, affiche `Yes`.
True/False	Affiche `False` si le nombre est 0 ; sinon, affiche `True`.
On/Off	Affiche `Off` si le nombre est 0 ; sinon, affiche `On`.

Notes du développeur sur Smart Device

Les formats **Yes/No**, **True/False** et **On/Off** ne sont pas pris en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Liste des conversions
Format, fonction
Formats de date/heure prédéfinis (fonction Format)
Formats numériques définis par l'utilisateur (fonction Format)

Formats de date/heure prédéfinis (fonction Format)

Le tableau suivant identifie les noms de formats de date et d'heure prédéfinis. Ceux-ci peuvent être utilisés par nom comme argument de style pour la fonction **Format** :

Nom de format	Description
General Date ou **G**	Affiche une date et/ou une heure. Par exemple, `4/3/93 05:34 PM`. L'affichage de la date est déterminé par la valeur de la culture en cours de votre système.
Long Date, **Medium Date** ou **D**	Affiche une date en fonction du format de date longue de la culture actuelle.
Short Date ou **d**	Affiche une date en utilisant le format de date courte de la culture actuelle.
Long Time, **Medium Time** ou **T**	Affiche une heure en utilisant le format d'heure longue de la culture actuelle ; comprend généralement les heures, minutes, secondes.
Short Time ou **t**	Affiche une heure en utilisant le format d'heure courte de la culture actuelle.
f	Affiche la date longue et l'heure courte en fonction du format de la culture actuelle.
F	Affiche la date longue et l'heure longue en fonction du format de la culture actuelle.
g	Affiche la date courte et l'heure courte en fonction du format de la culture en cours.
M, **m**	Affiche le mois et le jour d'une date.
R, **r**	Met en forme la date et l'heure à l'heure GMT.
s	Met en forme la date et l'heure sous la forme d'un index pouvant être trié.
u	Met en forme la date et l'heure sous la forme d'un index GMT pouvant être trié.
U	Met en forme la date et l'heure avec la date longue et l'heure longue en heure GMT.
Y, **y**	Met en forme la date sous la forme année et mois.

Pour plus d'informations sur les informations de culture actuelles de l'application, consultez Comment les informations de culture affectent les chaînes dans Visual Basic.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Liste des conversions
Format, fonction
Formats numériques prédéfinis (fonction Format)
Formats de date/heure définis par l'utilisateur (fonction Format)

Formats de date/heure définis par l'utilisateur (fonction Format)

Le tableau suivant montre les caractères que vous pouvez utiliser pour créer des formats de date/heure définis par l'utilisateur : Ces caractères de format respectent la casse, alors que ça n'était pas le cas dans les versions précédentes de Visual Basic.

Caractère	Description
(**:**)	Séparateur d'heure. Dans certains paramètres régionaux, d'autres caractères peuvent être utilisés pour représenter le séparateur d'heure. Le séparateur d'heure sépare les heures, les minutes et les secondes lorsque les valeurs d'heure sont mises en forme. Le caractère réel utilisé comme séparateur d'heure dans le résultat mis en forme est déterminé par la valeur de la culture actuelle de votre application.
(**/**)	Séparateur de date. Dans certains paramètres régionaux, d'autres caractères peuvent être utilisés pour représenter le séparateur de date. Le séparateur de date sépare le jour, le mois et l'année lorsque les valeurs de date sont mises en forme. Le caractère réel utilisé comme séparateur de date dans le résultat mis en forme est déterminé par la culture actuelle de votre application.
(**%**)	Utilisé pour indiquer que le caractère suivant doit être lu comme un format d'une seule lettre sans tenir compte des lettres de fin. Utilisé également pour indiquer qu'un format d'une seule lettre est lu comme un format défini par l'utilisateur. Voir ci-dessous pour plus de détails.
d	Affiche le jour sous la forme d'un nombre sans zéro non significatif (par exemple, `1`). Utilisez **%d** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
dd	Affiche le jour sous la forme d'un nombre avec un zéro non significatif (par exemple, `01`).
ddd	Affiche le jour sous la forme d'une abréviation (par exemple, `Sun`).
dddd	Affiche le jour sous la forme d'un nom complet (par exemple, `Sunday`).
M	Affiche le mois sous la forme d'un nombre sans zéro non significatif (par exemple, janvier correspond à `1`). Utilisez **%M** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
MM	Affiche le mois sous la forme d'un nombre avec un zéro non significatif (par exemple, `01/12/01`).
MMM	Affiche le mois sous la forme d'une abréviation (par exemple, `Jan`).
MMMM	Affiche le mois sous la forme d'un nom de mois complet (par exemple, `January`).

g **g**	Affiche la chaîne de période/ère (par exemple, `A.D.`)
h	Affiche l'heure sous la forme d'un nombre sans zéros non significatifs en utilisant l'horloge au format 12 heures (par exemple, `1:15:15 PM`). Utilisez **%h** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
h **h**	Affiche l'heure sous la forme d'un nombre avec des zéros non significatifs en utilisant l'horloge au format 12 heures (par exemple, `01:15:15 PM`).
H	Affiche l'heure sous la forme d'un nombre sans zéros non significatifs en utilisant l'horloge au format 24 heures (par exemple, `1:15:15`). Utilisez **%H** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
H **H**	Affiche l'heure sous la forme d'un nombre avec des zéros non significatifs en utilisant l'horloge au format 24 heures (par exemple, `01:15:15`).
m	Affiche les minutes sous la forme d'un nombre sans zéros non significatifs (par exemple, `12:1:15`). Utilisez **%m** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
m **m**	Affiche les minutes sous la forme d'un nombre avec des zéros non significatifs (par exemple, `12:01:15`).
s	Affiche les secondes sous la forme d'un nombre sans zéros non significatifs (par exemple, `12:15:5`). Utilisez **%s** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
ss	Affiche les secondes sous la forme d'un nombre avec des zéros non significatifs (par exemple, `12:15:05`).
f	Affiche les fractions de secondes. Par exemple, **ff** affiche les centièmes de secondes, tandis que **ffff** affiche les dizaines de millièmes de secondes. Vous pouvez utiliser jusqu'à 7 symboles **f** dans un format défini par l'utilisateur. Utilisez **%f** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
t	Utilise l'horloge au format 12 heures et affiche la lettre `A` majuscule pour toutes les heures avant midi ; affiche la lettre `P` majuscule pour toutes les heures entre midi et 11:59 P.M. Utilisez **%t** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
tt	Pour les paramètres régionaux qui utilisent l'horloge au format 12 heures, affiche `AM` en majuscules pour toutes les heures avant midi ; affiche `PM` en majuscules pour toutes les heures entre midi et 11:59 P.M. Pour les paramètres régionaux qui utilisent l'horloge au format 24 heures, n'affiche rien.
y	Affiche l'année sous la forme d'un nombre (0-9) sans zéros non significatifs. Utilisez **%y** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
yy	Affiche l'année sous un format numérique à deux chiffres avec un zéro non significatif, le cas échéant.
yy **y**	Affiche l'année sous un format numérique à quatre chiffres.
yy **yy**	Affiche l'année sous un format numérique à quatre chiffres.
z	Affiche le décalage horaire sous la forme d'un nombre sans zéro non significatif (par exemple, `-8`). Utilisez **%z** s'il s'agit du seul caractère dans le format numérique défini par l'utilisateur.
zz	Affiche le décalage horaire sous la forme d'un nombre avec un zéro non significatif (par exemple, `-08`).

zzz	Affiche le décalage horaire complet (par exemple, `-08:00`).

Pour plus d'informations sur les informations de culture actuelles de l'application, consultez Comment les informations de culture affectent les chaînes dans Visual Basic.

Exemple

Voici des exemples de formats de date et d'heure définis par l'utilisateur pour `December 7, 1958, 8:50 PM, 35 seconds`

Format	Affiche
`M/d/yy`	`12/7/58`
`d-MMM`	`7-Dec`
`d-MMMM-yy`	`7-December-58`
`d MMMM`	`7 December`
`MMMM yy`	`December 58`
`hh:mm tt`	`08:50 PM`
`h:mm:ss t`	`8:50:35 P`
`H:mm`	`20:50`
`H:mm:ss`	`20:50:35`
`M/d/yyyy H:mm`	`12/7/1958 20:50`

Notes du développeur sur Smart Device

La résolution de l'heure minimale pour un périphérique est déterminée par le fabricant du périphérique. Si la résolution d'heure pour le périphérique est assez approximative, le caractère de mise en forme **f** retourne 0 en cas d'exécution sur ce périphérique.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Liste des conversions
Format, fonction
Formats de date/heure prédéfinis (fonction Format)
Formats numériques définis par l'utilisateur (fonction Format)

Formats numériques définis par l'utilisateur (fonction Format)

Le tableau suivant identifie les caractères que vous pouvez utiliser pour créer des formats de nombres définis par l'utilisateur. Ceux-ci peuvent être utilisés pour créer l'argument *Style* pour la fonction **Format** :

Caractère	Description
Aucun	Affiche le nombre sans mise en forme.
(**0**)	Espace réservé à un chiffre. Affiche un chiffre ou un zéro. Affiche un chiffre si l'expression a un chiffre dans la position où le zéro apparaît dans la chaîne de format ; sinon, affiche un zéro dans cette position. Si le nombre contient moins de chiffres que de zéros (de part et d'autre du séparateur décimal) dans l'expression de format, affiche des zéros non significatifs ou à droite. Si le nombre contient davantage de chiffres à droite du séparateur décimal qu'il n'y a de zéros à droite du séparateur décimal dans l'expression de format, arrondit le nombre à autant de positions décimales qu'il y a de zéros. Si le nombre contient davantage de chiffres à gauche du séparateur décimal qu'il n'y a de zéros à gauche du séparateur décimal dans l'expression de format, affiche les chiffres supplémentaires sans modification.
(**#**)	Espace réservé à un chiffre. Affiche un chiffre ou rien. Affiche un chiffre si l'expression a un chiffre dans la position où le caractère **#** apparaît dans la chaîne de format ; sinon, n'affiche rien dans cette position. Ce symbole fonctionne comme l'espace réservé au **0**, sauf que les zéros non significatifs et à droite ne s'affichent pas si le nombre contient moins de chiffres qu'il n'y a de caractères **#** de part et d'autre du séparateur décimal dans l'expression de format.
(**.**)	Espace réservé aux décimales. L'espace réservé à la décimale détermine le nombre de chiffres affichés à gauche et à droite du séparateur décimal. Si l'expression de format ne contient que des caractères **#** à gauche de ce symbole, les nombres inférieurs à 1 commencent par un séparateur décimal. Pour afficher un zéro non significatif avec des nombres fractionnaires, utilisez zéro comme premier espace réservé à un chiffre à gauche du séparateur décimal. Dans certains paramètres régionaux, une virgule est utilisée comme séparateur décimal. Le caractère réel utilisé comme espace réservé à la décimale dans le résultat mis en forme dépend du format de nombre reconnu par votre système. Vous devriez donc utiliser le point comme espace réservé à la décimale dans vos mises en forme, même si vos paramètres régionaux utilisent la virgule à cette fin. La chaîne mise en forme apparaîtra dans le format correct pour les paramètres régionaux.
(**%**)	Espace réservé à un pourcentage. Multiplie l'expression par 100. Le caractère du pourcentage (**%**) est inséré dans la position où il apparaît dans la chaîne de format.

(**,**)	Séparateur de milliers. Le séparateur de milliers sépare les milliers des centaines dans un nombre qui possède quatre chiffres ou plus à gauche du séparateur décimal. L'utilisation standard du séparateur de milliers est spécifiée si le format contient un séparateur de milliers entouré par des espaces réservés aux chiffres (**0** ou **#**). Un séparateur de milliers placé immédiatement à gauche du séparateur décimal (qu'une décimale soit spécifiée ou non) ou en tant que caractère le plus à droite dans la chaîne, signifie « mettre à l'échelle le nombre en le divisant par 1 000, en l'arrondissant si nécessaire ». Les nombres plus petits que 1 000 mais supérieurs ou égaux à 500 sont affichés comme `1`, et les nombres plus petits que 500 sont affichés comme `0`. Deux séparateurs de milliers adjacents dans cette position ont pour facteur d'échelle 1 million et un facteur complémentaire de 1 000 pour chaque séparateur supplémentaire. Les séparateurs multiples dans n'importe quelle position autre que celle immédiatement à gauche du séparateur décimal ou la position la plus à droite dans la chaîne sont traités simplement comme le spécifie l'utilisation d'un séparateur de milliers. Dans certains paramètres régionaux, un point est utilisé comme séparateur de milliers. Le caractère réel utilisé comme séparateur de milliers dans le résultat mis en forme dépend du format de nombre reconnu par votre système. Vous devriez donc utiliser la virgule comme séparateur de milliers dans vos mises en forme, même si vos paramètres régionaux utilisent le point à cette fin. La chaîne mise en forme apparaîtra dans le format correct pour les paramètres régionaux. Par exemple, considérez les trois chaînes de format suivantes : • `"#,0."`, qui utilise le séparateur des milliers pour mettre en forme le nombre 100 millions comme chaîne "100 000 000". • `"#0,."`, qui utilise un facteur d'échelle égal à mille pour mettre en forme le nombre 100 millions comme chaîne "100000". • `"#,0,."`, qui utilise le séparateur de milliers et un facteur d'échelle égal à mille pour mettre en forme le nombre 100 millions comme chaîne "100 000".
(**:**)	Séparateur d'heure. Dans certains paramètres régionaux, d'autres caractères peuvent être utilisés pour représenter le séparateur d'heure. Le séparateur d'heure sépare les heures, les minutes et les secondes lorsque les valeurs d'heure sont mises en forme. Le caractère réel utilisé comme séparateur d'heure dans le résultat mis en forme est déterminé par les paramètres de votre système.
(**/**)	Séparateur de date. Dans certains paramètres régionaux, d'autres caractères peuvent être utilisés pour représenter le séparateur de date. Le séparateur de date sépare le jour, le mois et l'année lorsque les valeurs de date sont mises en forme. Le caractère réel utilisé comme séparateur de date dans le résultat mis en forme est déterminé par les paramètres de votre système.
(**E-** **E+** **e-** **e+**)	Format scientifique. Si l'expression de format contient au moins un espace réservé au chiffre (**0** ou **#**) à gauche de **E-**, **E+**, **e-** ou **e+**, le nombre est affiché au format scientifique et `E` ou `e` est inséré entre le nombre et son exposant. Le nombre d'espaces réservés aux chiffres à gauche détermine le nombre de chiffres dans l'exposant. Utilisez **E-** ou **e-** pour placer un signe moins à côté des exposants négatifs. Utilisez **E+** ou **e+** pour placer un signe moins à côté des exposants négatifs et un signe plus à côté des exposants positifs. Vous devez également inclure des espaces réservés aux chiffres à droite de ce symbole pour obtenir une mise en forme correcte.
- + $ ( )	Caractères littéraux. Ces caractères sont affichés exactement tels qu'ils sont tapés dans la chaîne de format. Pour afficher un caractère autre que ceux répertoriés, faites-le précéder d'une barre oblique inverse (****) ou placez-le entre guillemets (" ").
(****)	Affiche le caractère suivant dans la chaîne de format. Pour afficher un caractère ayant une signification spéciale en tant que caractère littéral, faites-le précéder d'une barre oblique inverse (****). La barre oblique inverse elle-même n'est pas affichée. L'utilisation d'une barre oblique inverse revient à placer le caractère suivant entre guillemets. Pour afficher une barre oblique inverse, utilisez deux barres obliques inverses (****). Parmi les caractères qui ne peuvent être affichés en tant que caractères littéraux, on peut citer les caractères de mise en forme de date et d'heure (**a**, **c**, **d**, **h**, **m**, **n**, **p**, **q**, **s**, **t**, **w**, **y**, **/** et **:**), les caractères de mise en forme numérique (**#**, **0**, **%**, **E**, **e**, virgule et point) et les caractères de mise en forme de chaînes (**@**, **&**, **<**, **>** et **!**).

("*ABC*")	Affiche la chaîne en la plaçant entre guillemets (" "). Pour inclure une chaîne dans l'argument de style à partir du code, vous devez utiliser **Chr(34)** pour inclure le texte (**34** est le code de caractère pour un guillemet (")).

Exemple

Le tableau suivant contient des exemples d'expressions de format pour des nombres (ces exemples supposent tous que votre système est défini avec les paramètres régionaux Anglais-États-Unis). La première colonne contient les chaînes de format pour l'argument *Style* de la fonction **Format** ; les autres colonnes contiennent le résultat obtenu si les données mises en forme ont la valeur donnée dans les en-têtes de colonnes.

Format (*Style*)	**"5" mis en forme comme suit**	**"-5" mis en forme comme suit**	**"0.5" mis en forme comme suit**
`Zero-length string ("")`	`5`	`-5`	`0.5`
`0`	`5`	`-5`	`1`
`0.00`	`5.00`	`-5.00`	`0.50`
`#,##0`	`5`	`-5`	`1`
`$#,##0;($#,##0)`	`$5`	`($5)`	`$1`
`$#,##0.00;($#,##0.00)`	`$5.00`	`($5.00)`	`$0.50`
`0%`	`500%`	`-500%`	`50%`
`0.00%`	`500.00%`	`-500.00%`	`50.00%`
`0.00E+00`	`5.00E+00`	`-5.00E+00`	`5.00E-01`
`0.00E-00`	`5.00E00`	`-5.00E00`	`5.00E-01`

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Liste des conversions
Formats différents pour valeurs numériques différentes (fonction Format)
Format, fonction
Formats numériques prédéfinis (fonction Format)
Formats de date/heure définis par l'utilisateur (fonction Format)

FormatCurrency, fonction (Visual Basic)

Retourne une expression sous forme de valeur monétaire utilisant le symbole monétaire défini dans le Panneau de configuration du système.

```
Function FormatCurrency(
   ByVal Expression As Object,
   Optional ByVal NumDigitsAfterDecimal As Integer = -1,
   Optional ByVal IncludeLeadingDigit As TriState = TriState.UseDefault,
   Optional ByVal UseParensForNegativeNumbers As TriState = TriState.UseDefault,
   Optional ByVal GroupDigits As TriState = TriState.UseDefault
) As String
```

Paramètres

Expression

Requis. Expression à formater.

NumDigitsAfterDecimal

Facultatif. Valeur numérique indiquant le nombre de décimales affichées. La valeur par défaut est –1, ce qui indique que les paramètres régionaux de l'ordinateur sont utilisés.

IncludeLeadingDigit

Facultatif. Énumération TriState (3 états) indiquant si un zéro non significatif s'affiche pour les valeurs décimales. Consultez la section « Paramètres » pour obtenir les valeurs.

UseParensForNegativeNumbers

Facultatif. Énumération TriState (3 états) indiquant si les valeurs négatives doivent être mises entre parenthèses. Consultez la section « Paramètres » pour obtenir les valeurs.

GroupDigits

Facultatif. Énumération TriState (3 états) indiquant si les nombres doivent ou non être regroupés en utilisant le délimiteur de groupe spécifié dans les paramètres régionaux de l'ordinateur. Consultez la section « Paramètres » pour obtenir les valeurs.

Paramètres

Les arguments *IncludeLeadingDigit*, *UseParensForNegativeNumbers* et *GroupDigits* prennent les valeurs d'énumération TriState suivantes.

Valeur	Description
TriState.True	True
TriState.False	False
TriState.UseDefault	Paramètres régionaux de l'ordinateur

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Le nombre de chiffres après la virgule est supérieur à 99.
InvalidCastException	13	Le type n'est pas numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Quand un ou plusieurs arguments facultatifs sont omis, les valeurs sont fournies par les paramètres régionaux correspondants de l'ordinateur.

La position du symbole monétaire par rapport à la valeur monétaire est déterminée par les paramètres régionaux du système.

Remarque

Tous les paramètres sont issus des paramètres régionaux de l'application. Par défaut, il s'agit des paramètres régionaux définis dans le Panneau de configuration. Toutefois, ceux-ci peuvent être modifiés par programme à l'aide du .NET Framework, à l'exception du zéro non significatif, qui est un paramètre défini dans l'onglet **Number**.

Exemple

L'exemple ci-dessous illustre l'utilisation de la fonction **FormatCurrency**.

VB

```
Dim TestDebt As Double = -4456.43
Dim TestString As String
' Returns "($4,456.43)".
TestString = FormatCurrency(TestDebt, , , TriState.True, TriState.True)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
FormatDateTime, fonction (Visual Basic)
FormatNumber, fonction (Visual Basic)
FormatPercent, fonction (Visual Basic)
Tristate, énumération
ArgumentException
InvalidCastException

FormatDateTime, fonction (Visual Basic)

Retourne une expression de chaîne représentant une valeur de date/d'heure.

```
Function FormatDateTime(
   ByVal Expression As DateTime,
   Optional ByVal NamedFormat As DateFormat = DateFormat.GeneralDate
) As String
```

Paramètres

Expression

Requis. Expression **Date** à mettre en forme.

NamedFormat

Facultatif. Valeur numérique qui indique le format de date/d'heure utilisé. Si cette valeur est omise, **DateFormat.GeneralDate** est utilisé.

Paramètres

L'argument *NamedFormat* possède les paramètres suivants :

Constante	Description
DateFormat.GeneralDate	Affiche une date et/ou une heure. Affiche une partie de date sous forme de date courte. En présence d'une partie de date, elle l'affiche sous forme d'heure complète. Si elles sont présentes, les deux parties sont affichées.
DateFormat.LongDate	Affiche une date au format de date longue spécifié dans les paramètres régionaux de votre ordinateur.
DateFormat.ShortDate	Affiche une date au format de date courte spécifié dans les paramètres régionaux de votre ordinateur.
DateFormat.LongTime	Affiche une heure au format d'heure spécifié dans les paramètres régionaux de votre ordinateur.
DateFormat.ShortTime	Affiche une heure au format 24 heures (hh:mm).

Exceptions

Type d'exception	Numéro d'erreur	Condition
ArgumentException	5	Le paramètre *NamedFormat* n'est pas valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Le type de données **Date** contient toujours des informations sur la date et l'heure. Dans le cadre de la conversion de type, Visual Basic considère la valeur 1/1/1 (le 1er janvier de l'année 1) comme une valeur neutre pour la date et 00:00:00 (minuit) comme une valeur neutre pour l'heure. Si vous formatez une valeur **Date** en chaîne de date/d'heure, **FormatDateTime** n'inclut pas de valeurs neutres dans la chaîne résultante. Par exemple, si vous convertissez #1/1/0001 9:30:00# en chaîne, le résultat est "9:30:00 AM" ; les informations de date ayant été supprimées. Cependant, les informations de date se trouvent toujours dans la valeur **Date** d'origine et peuvent être récupérées à l'aide de fonctions telles que **DatePart**.

Remarque

Si vous passez l'argument *Expression* comme un littéral **String**, **FormatDateTime** l'interprète en fonction du paramètre CurrentCulture de votre application. Cependant, si vous le passez comme un littéral **Date**, vous devez utiliser le format #mm /dd/yyyy#, car **FormatDateTime** interprète toujours un littéral de **Date** en fonction de la culture anglo-américaine. Cela est nécessaire, car si une application est développée et codée à l'aide de littéraux **Date** provenant d'une certaine culture, puis exécutée sur une plate-forme d'une autre culture, les littéraux **Date** peuvent être mal analysés.

Exemple

L'exemple ci-dessous illustre l'utilisation de la fonction **FormatDateTime** :

VB

```
' English (US) format.
Dim TestDate As DateTime = #3/12/1999#

' FormatDateTime returns "Friday, March 12, 1999".
' The time information is neutral (00:00:00) and therefore suppressed.
Dim TestString As String = FormatDateTime(TestDate, DateFormat.LongDate)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FormatCurrency, fonction (Visual Basic)
FormatNumber, fonction (Visual Basic)
FormatPercent, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
Liste des manipulations de chaînes
ArgumentException

FormatNumber, fonction (Visual Basic)

Retourne une expression sous forme de nombre.

```
Function FormatNumber(
   ByVal Expression As Object,
   Optional ByVal NumDigitsAfterDecimal As Integer = -1,
   Optional ByVal IncludeLeadingDigit As TriState = TriState.UseDefault,
   Optional ByVal UseParensForNegativeNumbers As TriState = TriState.UseDefault,
   Optional ByVal GroupDigits As TriState = TriState.UseDefault
) As String
```

Paramètres

Expression

Requis. Expression à formater.

NumDigitsAfterDecimal

Facultatif. Valeur numérique indiquant le nombre de décimales affichées. La valeur par défaut est –1, ce qui indique que les paramètres régionaux de l'ordinateur sont utilisés.

IncludeLeadingDigit

Facultatif. Constante Tristate (3 états) qui indique si un 0 non significatif s'affiche pour les valeurs décimales. Consultez « Settings » pour obtenir les valeurs.

UseParensForNegativeNumbers

Facultatif. Constante Tristate (3 états) qui indique si les valeurs négatives doivent être mises entre parenthèses. Consultez « Settings » pour obtenir les valeurs.

GroupDigits

Facultatif. Constante Tristate (3 états) qui indique si les nombres doivent ou non être regroupés en utilisant le délimiteur de groupe spécifié dans les paramètres régionaux de l'ordinateur. Consultez « Settings » pour obtenir les valeurs.

Paramètres

Les arguments *IncludeLeadingDigit*, *UseParensForNegativeNumbers* et *GroupDigits* prennent les valeurs suivantes.

Constante	Description
TriState.True	True
TriState.False	False
TriState.UseDefault	Paramètres régionaux de l'ordinateur

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	13	Le type n'est pas numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Lorsqu'un ou plusieurs des arguments facultatifs sont omis, les valeurs de ces arguments sont fournies par les paramètres régionaux.

Remarque

Tous les paramètres sont issus des paramètres régionaux de l'application. Par défaut, il s'agit des paramètres régionaux définis dans le Panneau de configuration. Toutefois, ceux-ci peuvent être modifiés par programme à l'aide du .NET Framework.

Exemple

Cet exemple illustre la fonction **FormatNumber**.

VB

```
Dim TestNumber As Integer = 45600
' Returns "45,600.00".
Dim TestString As String = FormatNumber(TestNumber, 2, , , TriState.True)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
FormatCurrency, fonction (Visual Basic)
FormatDateTime, fonction (Visual Basic)
FormatPercent, fonction (Visual Basic)
Tristate, énumération
InvalidCastException

FormatPercent, fonction (Visual Basic)

Retourne une expression formatée sous forme de pourcentage (c'est-à-dire multipliée par 100) avec un caractère de fin %.

```
Function FormatPercent(
   ByVal Expression As Object,
   Optional ByVal NumDigitsAfterDecimal As Integer = -1,
   Optional ByVal IncludeLeadingDigit As TriState = TriState.UseDefault,
   Optional ByVal UseParensForNegativeNumbers As TriState = TriState.UseDefault,
   Optional ByVal GroupDigits As TriState = TriState.UseDefault
) As String
```

Paramètres

Expression

Requis. Expression à formater.

NumDigitsAfterDecimal

Facultatif. Valeur numérique indiquant le nombre de décimales affichées. La valeur par défaut est –1, ce qui indique que les paramètres régionaux sont utilisés.

IncludeLeadingDigit

Facultatif. Constante **TriState** qui indique si un zéro non significatif s'affiche pour les valeurs décimales. Consultez « Settings » pour obtenir les valeurs.

UseParensForNegativeNumbers

Facultatif. Constante **TriState** qui indique si les valeurs négatives doivent être placées entre parenthèses. Consultez « Settings » pour obtenir les valeurs.

GroupDigits

Facultatif. Constante **TriState** qui indique si les nombres doivent ou non être regroupés en utilisant le délimiteur de groupe spécifié dans les paramètres régionaux de l'ordinateur. Consultez « Settings » pour obtenir les valeurs.

Paramètres

Les arguments *IncludeLeadingDigit*, *UseParensForNegativeNumbers* et *GroupDigits* prennent les valeurs suivantes.

Constante	Description
TriState.True	True
TriState.False	False
TriState.Default	Paramètres régionaux de l'ordinateur

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	13	Le type n'est pas numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Lorsqu'un ou plusieurs des arguments facultatifs sont omis, les valeurs de ces arguments sont fournies par les paramètres régionaux.

Remarque

Tous les paramètres sont issus des paramètres régionaux de l'application. Par défaut, il s'agit des paramètres régionaux définis dans le Panneau de configuration. Toutefois, ceux-ci peuvent être modifiés par programme à l'aide du .NET Framework.

Exemple

L'exemple ci-dessous illustre l'utilisation de la fonction **FormatPercent** :

VB

```
Dim TestNumber As Single = 0.76
' Returns "76.00%".
Dim TestString As String = FormatPercent(TestNumber)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
FormatCurrency, fonction (Visual Basic)
FormatDateTime, fonction (Visual Basic)
FormatNumber, fonction (Visual Basic)
Tristate, énumération
InvalidCastException

FreeFile, fonction

Retourne une valeur de type **Integer** représentant le prochain numéro de fichier disponible pouvant être utilisé par la fonction **FileOpen**.

```
Public Function FreeFile() As Integer
```

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	67	Plus de 255 fichiers sont en cours d'utilisation.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **FreeFile** pour fournir un numéro de fichier encore non utilisé.

Exemple

Cet exemple utilise la fonction **FreeFile** pour retourner le prochain numéro de fichier disponible. Cinq fichiers sont ouverts à l'intérieur de la boucle et des données exemple sont écrites dans chacun d'eux.

VB

```
Dim count As Integer
Dim fileNumber As Integer
For count = 1 To 5
   fileNumber = FreeFile()
   FileOpen(fileNumber, "TEST" & count & ".TXT", OpenMode.Output)
   PrintLine(fileNumber, "This is a sample.")
   FileClose(fileNumber)
Next
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileOpen, fonction

IOException Class

Autres ressources

Écriture dans des fichiers en Visual Basic

FV, fonction

Retourne une valeur de type **Double** indiquant le futur montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.

```
Function FV( _
   ByVal Rate As Double, _
   ByVal NPer As Double, _
   ByVal Pmt As Double, _
   Optional ByVal PV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

Paramètres

Rate

Requis. **Double** spécifiant le taux d'intérêt par période. Par exemple, si vous obtenez un prêt pour l'achat d'une voiture à un taux d'intérêt annuel (TIA) de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

NPer

Requis. **Double** spécifiant le nombre total d'échéances de l'annuité. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt d'une durée de quatre ans pour l'achat d'une voiture, votre emprunt comporte un total de 4 x 12 (soit 48) échéances.

Pmt

Requis. **Double** spécifiant le versement à effectuer à chaque échéance. Les paiements comprennent généralement le capital et les intérêts qui ne sont pas modifiés au cours de l'annuité.

PV

Facultatif. Donnée de type **Double** indiquant la valeur actuelle (ou montant forfaitaire) d'une série de paiements futurs. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond pour le prêteur à la valeur actuelle des remboursements mensuels que vous allez effectuer pour la voiture. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être **DueDate.EndOfPeriod** si les paiements sont dus à terme échu ou **DueDate.BegOfPeriod** si les paiements sont dus à terme à échoir. Si cet argument est omis, **DueDate.EndOfPeriod** est utilisé par défaut.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Les arguments *Rate* et *NPer* doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités. Par exemple, si *Rate* est exprimé en mois, il doit en être de même pour *NPer*.

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Exemple

Cet exemple utilise la fonction **FV** pour retourner la valeur future d'un investissement, en fonction des intérêts produits par période (`APR / 12`), du nombre total des paiements (`TotPmts`), du montant des versements (`Payment`), de la valeur actuelle de l'investissement (`PVal`) et d'un nombre indiquant si les paiements s'effectuent au début ou à la fin des échéances (`PayType`). Notez que `Payment` doit être une valeur négative puisqu'il s'agit d'une sortie de trésorerie.

VB

```
Sub TestFV()
    Dim TotPmts As Integer
```

```
    Dim Payment, APR, PVal, Fval As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    Payment = CDbl(InputBox("How much do you plan to save each month?"))
    APR = CDbl(InputBox("Enter the expected interest annual percentage rate."))
    ' Ensure proper form.
    If APR > 1 Then APR = APR / 100
    TotPmts = CInt(InputBox("For how many months do you expect to save?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    PVal = CDbl(InputBox("How much is in this savings account now?"))
    Fval = FV(APR / 12, TotPmts, -Payment, -PVal, PayType)
    MsgBox("Your savings will be worth " & Format(Fval, Fmt) & ".")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

GetAllSettings, fonction

Retourne une liste de paramètres de clé et leurs valeurs respectives (créées à l'origine avec **SaveSetting**) pour une application dans la base de registres de Windows.

L'utilisation de la fonctionnalité **My** vous permet une meilleure productivité et de meilleures performances dans les opérations de Registre que **GetAllSettings**. Pour plus d'informations, consultez My.Computer.Registry, objet.

```
Public Function GetAllSettings( _
   ByVal AppName As String, _
   ByVal Section As String _
) As String(,)
```

Paramètres

AppName

Requis. Expression **String** contenant le nom de l'application ou du projet dont les paramètres de clé sont demandés.

Section

Requis. Expression **String** qui contient le nom de la section dont les paramètres de clé sont demandés. **GetAllSettings** retourne un objet qui contient un tableau de chaînes à deux dimensions. Les chaînes contiennent tous les paramètres de clé dans la section spécifiée ainsi que leurs valeurs correspondantes.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	L'utilisateur n'est pas connecté.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

GetAllSettings retourne un **Object** non initialisé si *AppName* ou *Section* n'existe pas.

GetAllSettings nécessite qu'un utilisateur soit connecté, car elle fonctionne sous la clé de Registre **HKEY_LOCAL_USER**, qui n'est active que lorsque l'utilisateur se connecte de façon interactive.

Les paramètres de Registre accessibles via un processus non interactif (tel que Mtx.exe) doivent être stockés au-dessous des clés de Registre **HKEY_LOCAL_MACHINE\Software** ou **HKEY_USER\DEFAULT\Software**.

Exemple

Cet exemple utilise d'abord la fonction **SaveSetting** pour entrer des données dans la base de registres de Windows pour l'application spécifiée en tant que *AppName*, puis utilise la fonction **GetAllSettings** pour afficher les paramètres. Notez que les noms d'applications et les noms *Section* ne peuvent pas être récupérés avec **GetAllSettings**. Enfin, la fonction **DeleteSetting** supprime les entrées de l'application.

VB

```
' Object to hold 2-dimensional array returned by GetAllSettings.
' Integer to hold counter.
Dim MySettings(,) As String
Dim intSettings As Integer
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp", "Startup", "Left", "50")
' Retrieve the settings.
MySettings = GetAllSettings("MyApp", "Startup")
For intSettings = LBound(MySettings, 1) To UBound(MySettings, 1)
   WriteLine(1, MySettings(intSettings, 0))
   WriteLine(1, MySettings(intSettings, 1))
```

```
    Next intSettings
    DeleteSetting("MyApp")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DeleteSetting, fonction
GetSetting, fonction
SaveSetting, fonction
ArgumentException Class

Concepts

Tâches courantes relatives au Registre

GetAttr, fonction

Retourne une valeur **FileAttribute** représentant les attributs d'un fichier, d'un répertoire ou d'un dossier.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **FileAttribute**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Function GetAttr(ByVal PathName As String) As FileAttribute
```

Paramètres

PathName

Requis. Expression **String** qui spécifie un nom de fichier, de répertoire ou de dossier. *PathName* peut inclure le répertoire ou dossier ainsi que le lecteur.

Valeur de retour

La valeur retournée par **GetAttr** est la somme des valeurs d'énumération suivantes :

Valeur	Constante	Description
Normal	**vbNormal**	Normal.
ReadOnly	**vbReadOnly**	Lecture seule.
Hidden	**vbHidden**	Caché.
System	**vbSystem**	Fichier système.
Directory	**vbDirectory**	Répertoire ou dossier.
Archive	**vbArchive**	Fichier modifié depuis la dernière sauvegarde.
Alias	**vbAlias**	Le fichier a un nom différent.

Remarque

Ces énumérations sont spécifiées par le langage Visual Basic. Les noms peuvent remplacer les valeurs réelles partout dans v otre code.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*Pathname* n'est pas valide ou contient des caractères génériques.
FileNotFoundException	53	Le fichier cible n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Pour déterminer les attributs définis, utilisez l'opérateur **And** pour effectuer une comparaison d'opérations de bits entre la valeur retournée par la fonction **GetAttr** et la valeur de l'attribut de fichier individuel souhaité. Si le résultat est différent de zéro, cet attribut est défini pour le fichier désigné. Par exemple, la valeur de retour de l'expression **And** suivante est zéro si l'attribut **Archive** n'est pas défini :

```
Result = GetAttr(FName) And vbArchive
```

Une valeur différente de zéro est retournée si l'attribut **Archive** est défini.

Exemple

Cet exemple utilise la fonction **GetAttr** pour déterminer les attributs d'un fichier et un répertoire ou un dossier.

VB

```
Dim MyAttr As FileAttribute
' Assume file TESTFILE is normal and readonly.
MyAttr = GetAttr("C:\TESTFILE.txt")   ' Returns vbNormal.

' Test for normal.
If (MyAttr And FileAttribute.Normal) = FileAttribute.Normal Then
   MsgBox("This file is normal.")
End If

' Test for normal and readonly.
Dim normalReadonly As FileAttribute
normalReadonly = FileAttribute.Normal Or FileAttribute.ReadOnly
If (MyAttr And normalReadonly) = normalReadonly Then
   MsgBox("This file is normal and readonly.")
End If

' Assume MYDIR is a directory or folder.
MyAttr = GetAttr("C:\MYDIR")
If (MyAttr And FileAttribute.Directory) = FileAttribute.Directory Then
   MsgBox("MYDIR is a directory")
End If
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

And, opérateur (Visual Basic)
FileAttr, fonction
SetAttr, fonction
IOException Class
FileNotFoundException Class
FileAttribute, énumération

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

GetChar, fonction

Retourne une valeur **Char** représentant le caractère issu de l'index spécifié dans la chaîne fournie.

```
Public Shared Function GetChar( _
   ByVal str As String, _
   ByVal Index As Integer _
) As Char
```

Paramètres

str

Requis. Toute expression **String** valide.

Index

Requis. Expression **Integer**. Index (de base 1) du caractère de *str* à retourner.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*str* est **Nothing**, *Index* est inférieur à 1 ou *Index* est supérieur à l'index du dernier caractère de *str*.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si *Index* est inférieur à 1 ou supérieur à l'index du dernier caractère de *str*, un **ArgumentException** est levé.

Exemple

L'exemple suivant montre comment utiliser la fonction **GetChar** pour retourner un caractère à partir d'un index spécifié dans une expression **String**.

VB

```
Dim TestString As String = "ABCDE"
Dim TestChar As Char
' Returns "D"
TestChar = GetChar(TestString, 4)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Mid, fonction (Visual Basic)
ArgumentException

GetException, fonction

Retourne l'exception représentant l'erreur qui s'est produite.

```
Overridable Public Function GetException() As Exception
```

Notes

La fonction **GetException** est disponible uniquement à partir de la classe de l'objet **Err**. Elle fonctionne avec la propriété **Exception** de l'objet **Err** afin d'afficher l'erreur qui s'est produite.

Exemple

Le code suivant affiche le message assigné à l'exception dans l'objet **Err** :

VB

```
On Error Resume Next
Dim myError As System.Exception
' Generate an overflow exception.
Err.Raise(6)
' Assigns the exception from the Err object to myError.
myError = Err.GetException()
' Displays the message associated with the exception.
MsgBox(myError.Message)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : Err Object

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : extraire des informations à partir d'un objet Error

Référence

Err, objet (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Vue d'ensemble de la gestion non structurée des exceptions

GetObject, fonction (Visual Basic)

Retourne une référence à un objet fourni par un composant COM.

```
Public Function GetObject( _
    Optional ByVal PathName As String = Nothing, _
    Optional ByVal [Class] As String = Nothing _
) As Object
```

Paramètres

Paramètres	Description
PathName	Facultatif. **String**. Chemin d'accès complet et nom du fichier contenant l'objet à récupérer. Si *PathName* est omis ou constitue une chaîne de longueur nulle (**""**), *Class* est obligatoire.
Class	Requis si *PathName* n'est pas fourni. **String**. Chaîne représentant la classe de l'objet. L'argument *Class* emploie la syntaxe et les paramètres suivants : *appname* . *objecttype* **Paramètres** / **Description** *appname* — Requis. **String**. Nom de l'application fournissant l'objet. *objecttype* — Requis. **String**. Type ou classe de l'objet à créer.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
Exception	429	Aucun objet du type spécifié n'existe.
FileNotFoundException	432	Aucun objet avec le chemin d'accès spécifié et le nom de fichier n'existe.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **GetObject** pour charger une instance d'un composant COM à partir d'un fichier. L'exemple suivant illustre ce comportement :

```
Dim CADObject As Object
CADObject = GetObject("C:\CAD\schema.cad")
```

Lorsque ce code est exécuté, l'application associée au *PathName* spécifié démarre et l'objet dans le fichier spécifié est activé.

Cas par défaut

Si *PathName* est une chaîne de longueur nulle (**""**), **GetObject** retourne une nouvelle instance d'objet du type de classe spécifié. Si l'argument *PathName* est omis, **GetObject** retourne un objet actuellement actif du type de classe spécifié dans *Class*. S'il n'existe aucun objet du type spécifié, une erreur se produit.

Accès à un sous-objet

Certaines applications vous permettent d'activer un sous-objet associé à un fichier. Pour ce faire, ajoutez un point d'exclamation (**!**) à la fin du nom du fichier et faites suivre ce dernier d'une chaîne identifiant la partie du fichier à activer. Pour plus d'informations sur la façon de créer cette chaîne, consultez la documentation de l'application qui a créé l'objet.

Dans une application de dessin, par exemple, un fichier peut contenir un dessin constitué de plusieurs couches. Le code suivant

vous permet d'activer une couche d'un dessin nommé `schema.cad`.

```
layerObject = GetObject("C:\CAD\schema.cad!Layer3")
```

Spécification d'une classe

Si vous ne spécifiez pas la *Class* de l'objet, Automation détermine l'application à démarrer et l'objet à activer, en fonction du nom de fichier que vous fournissez. Cependant, certains fichiers peuvent prendre en charge plusieurs classes d'objets. Par exemple, un dessin peut prendre en charge trois types d'objets différents : un objet `Application`, un objet `Drawing` et un objet `Toolbar`, chacun faisant partie du même fichier. Pour spécifier l'objet à activer dans un fichier, utilisez l'argument *Class* facultatif. L'exemple suivant illustre ce comportement :

```
Dim drawObj As Object
drawObj = GetObject("C:\Drawings\sample.drw", "Figment.Drawing")
```

Dans l'exemple précédent, `Figment` est le nom d'une application de dessin et `Drawing` est un des types d'objets qu'elle prend en charge.

Utilisation de l'objet

Une fois un objet activé, vous y faites référence dans le code à l'aide de la variable objet que vous avez déclarée. Dans l'exemple précédent, vous accédez aux propriétés et aux méthodes du nouvel objet en utilisant la variable objet `drawObj`. L'exemple suivant illustre ce comportement :

```
drawObj.Line(9, 90)
drawObj.InsertText(9, 100, "Hello, world.")
drawObj.SaveAs("C:\Drawings\sample.drw")
```

Remarque

Utilisez la fonction **GetObject** s'il existe une instance actuelle de l'objet, ou lorsque vous souhaitez créer l'objet avec un fichier chargé. S'il n'existe aucune instance actuelle et si vous ne souhaitez pas démarrer l'objet avec un fichier chargé, utilisez CreateObject, fonction (Visual Basic).

Si un objet s'est inscrit comme objet à instance unique Active-X, une seule instance de l'objet est créée quel que soit le nombre d'appels à **CreateObject**. Avec un objet à instance unique, **GetObject** retourne toujours la même instance lorsqu'elle est appelée à l'aide de la syntaxe de chaîne de longueur nulle (**""**). En outre, elle produit une erreur si l'argument *PathName* est omis. Vous ne pouvez pas utiliser **GetObject** pour obtenir une référence à une classe créée à l'aide de Visual Basic.

Remarque de sécurité

La fonction **GetObject** nécessite une permission de code non managée, qui peut affecter son exécution dans les situations de confiance partielle. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Lorsque vous avez fini d'utiliser l'objet, définissez toutes les références à l'objet avec la valeur **Nothing**. Cela permet au runtime de disposer du composant COM.

Exemple

L'exemple suivant utilise la fonction **GetObject** pour déterminer si Excel est en cours d'exécution.

VB

```
' Test to see if a copy of Excel is already running.
Private Sub testExcelRunning()
    On Error Resume Next
    ' GetObject called without the first argument returns a
    ' reference to an instance of the application. If the
    ' application is not already running, an error occurs.
    Dim excelObj As Object = GetObject(, "Excel.Application")
    If Err.Number = 0 Then
        MsgBox("Excel is running")
    Else
        MsgBox("Excel is not running")
```

```
    End If
    Err.Clear()
    excelObj = Nothing
End Sub
```

Cet exemple utilise la fonction **GetObject** pour obtenir une référence à une feuille de calcul Microsoft Excel spécifique (`excelObj`). L'appel à **GetObject** retourne une référence à la feuille de calcul représentée par le fichier spécifié, `test.xls`. L'exemple de code rend ensuite Excel et la fenêtre contenant la feuille de calcul spécifiée visibles.

Cet exemple requiert **Option Strict Off** parce qu'il utilise la liaison tardive, où les objets sont assignés aux variables de type **Object**. Vous pouvez spécifier **Option Strict On** et déclarer des objets de types d'objets spécifiques si vous ajoutez une référence de projet à la bibliothèque de types Excel. À cette fin, ouvrez la boîte de dialogue **Ajouter une référence** à partir du menu **Projet** de Visual Studio et sélectionnez la bibliothèque de types Excel sur l'onglet **COM**.

VB

```
' Add Option Strict Off to the top of your program.
Option Strict Off
```

VB

```
Private Sub getExcel()
    Dim fileName As String = "c:\vb\test.xls"

    If Not My.Computer.FileSystem.FileExists(fileName) Then
        MsgBox(fileName & " does not exist")
        Exit Sub
    End If

    ' Set the object variable to refer to the file you want to use.
    Dim excelObj As Object = GetObject(fileName)
    ' Show Excel through its Application property.
    excelObj.Application.Visible = True
    ' Show the window containing the file.
    Dim winCount As Integer = excelObj.Parent.Windows.Count()
    excelObj.Parent.Windows(winCount).Visible = True

    ' Insert additional code to manipulate the test.xls file here.
    ' ...

    excelObj = Nothing
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

CreateObject, fonction (Visual Basic)
Declare, instruction
Option Strict, instruction
Exception
FileNotFoundException

GetSetting, fonction

Retourne une valeur de paramètre de clé pour une application dans la base de registres de Windows.

La fonctionnalité **My** vous permet une meilleure productivité et de meilleures performances dans les opérations de Registre que **GetAllSettings**. Pour plus d'informations, consultez My.Computer.Registry, objet.

```
Public Function GetSetting( _
   ByVal AppName As String, _
   ByVal Section As String, _
   ByVal Key As String, _
   Optional ByVal Default As String = "" _
) As String
```

Paramètres

AppName

Requis. Expression **String** contenant le nom de l'application ou du projet dont le paramètre de clé est demandé.

Section

Requis. Expression **String** contenant le nom de la section dans laquelle le paramètre de clé est trouvé.

Key

Requis. Expression **String** contenant le nom du paramètre de clé à retourner.

Default

Facultatif. Expression contenant la valeur à retourner si aucune valeur n'est définie dans le paramètre *Key*. S'il est omis, la valeur *Default* est considérée comme une chaîne de longueur nulle ("").

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Au moins un argument n'est pas une expression **String** ou l'utilisateur n'est pas connecté.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si l'un des éléments nommés dans les arguments **GetSetting** n'existe pas, **GetSetting** retourne la valeur *Default*.

GetSetting nécessite qu'un utilisateur soit connecté, car elle fonctionne sous la clé de Registre **HKEY_LOCAL_USER**, qui n'est active que lorsque l'utilisateur se connecte de façon interactive.

Les paramètres de Registre accessibles via un processus non interactif (tel que Mtx.exe) doivent être stockés au-dessous des clés de Registre **HKEY_LOCAL_MACHINE\Software** ou **HKEY_USER\DEFAULT\Software**.

GetSetting nécessite l'autorisation Registre **Read** .

Exemple

Cet exemple utilise d'abord la fonction **SaveSetting** pour entrer des données dans la base de registres de Windows pour l'application spécifiée en tant que *AppName*, puis utilise la fonction **GetSetting** pour afficher l'un des paramètres. Dans la mesure où l'argument *Default* est spécifié, le retour d'une valeur est garanti. Notez que les noms de *Section* ne peuvent pas être récupérés par **GetSetting**. Enfin, la fonction **DeleteSetting** supprime les entrées de l'application.

VB

```
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
```

```
SaveSetting("MyApp", "Startup", "Left", "50")
Console.WriteLine(GetSetting("MyApp", "Startup", "Left", "25"))
DeleteSetting("MyApp")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DeleteSetting, fonction
GetAllSettings, fonction
SaveSetting, fonction
ArgumentException Class
RegistryPermission Class

Concepts

Tâches courantes relatives au Registre

Fonctions H-L

Le tableau suivant contient une liste des fonctions membres runtime Visual Basic.

Hex	Hour	IIf	Input
InputBox	InputString	InStr	InStrRev
Int	IPmt	IRR	IsArray
IsDate	IsDBNull	IsError	IsNothing
IsNumeric	IsReference	Join	Kill
LBound	LCase	Left	Len
LineInput	Loc	Lock	LOF
LSet	LTrim		

La fonction mathématique **Log**, qui était disponible en tant que fonction membre runtime dans les versions 6.0 et antérieures de Visual Basic, a été remplacée par une méthode (Log) de la classe Math de la bibliothèque de classes .NET Framework. Pour plus d'informations, consultez Fonctions mathématiques (Visual Basic).

Notes du développeur sur Smart Device

Les éléments suivants ne sont pas pris en charge dans les applications Smart Device : **Input**, **InputString**, **Kill**, **LineInput**, **Loc**, **Lock** et **LOF**.

Voir aussi

Référence

Fonctions A-C
Fonctions D-G
Fonctions M-R
Fonctions S-Z

Autres ressources

Référence Visual Basic

Hex, fonction (Visual Basic)

Retourne une chaîne représentant la valeur hexadécimale d'un nombre.

```
Public Shared Function Hex( _
   ByVal Number As { Byte | SByte | Short | UShort |
   Integer | UInteger | Long | ULong | Object } _
) As String
```

Paramètres

Number

Requis. Toute expression numérique ou expression **String** valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	5	*Number* n'est pas spécifié.
ArgumentException	5	*Number* n'est pas un type numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si *Number* n'est pas déjà un nombre entier, il est arrondi au nombre entier le plus proche avant d'être évalué.

Si *Number* a la valeur	Hex retourne
Empty	Zéro (0)
Toute valeur numérique	Seize caractères hexadécimaux maximum

Vous pouvez représenter des nombres hexadécimaux directement en faisant précéder les nombres de la plage appropriée de **&H**. Par exemple, `&H10` représente le nombre décimal 16 en notation hexadécimale.

Exemple

Cet exemple utilise la fonction **Hex** pour retourner la valeur hexadécimale d'un nombre.

VB

```
Dim TestHex As String
' Returns 5.
TestHex = Hex(5)
' Returns A.
TestHex = Hex(10)
' Returns 1CB.
TestHex = Hex(459)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : convertir des chaînes hexadécimales en nombres

Référence

Oct, fonction

Fonctions de conversion de types de données

ArgumentNullException

Hour, fonction (Visual Basic)

Retourne une valeur **Integer** comprise entre 0 et 23 représentant l'heure du jour.

```
Public Function Hour(ByVal TimeValue As DateTime) As Integer
```

Paramètres
TimeValue

Requis. Valeur **Date** de laquelle vous souhaitez extraire l'heure.

Notes

Vous pouvez également obtenir l'heure du jour en appelant **DatePart** et en spécifiant **DateInterval.Hour** pour l'argument *Interval*.

Exemple

Cet exemple utilise la fonction **Hour** pour obtenir l'heure à partir d'une heure spécifiée. Dans l'environnement de développement, le littéral d'heure est affiché sous le format d'heure abrégée en utilisant les paramètres régionaux de votre code.

VB

```
Dim someTime As Date
Dim someHour As Integer
someTime = #4:35:17 PM#
someHour = Hour(someTime)
' someHour now contains 16.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Day, fonction (Visual Basic)
Minute, fonction
Now, propriété
Second, fonction (Visual Basic)
TimeOfDay, propriété
DatePart, fonction (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

IIf, fonction

Retourne un ou deux objets selon l'évaluation d'une expression.

```
Public Function IIf( _
   ByVal Expression As Boolean, _
   ByVal TruePart As Object, _
   ByVal FalsePart As Object _
) As Object
```

Paramètres

Expression

Requis. **Boolean**. Expression que vous voulez évaluer.

TruePart

Requis. **Object**. Retourné si *Expression* a la valeur **True**.

FalsePart

Requis. **Object**. Retourné si *Expression* a la valeur **False**.

Notes

La fonction **IIf** fournit un équivalent pour le Conditional Operator: ? : ternaire dans Visual C++.

Exemple

Cet exemple utilise la fonction **IIf** pour évaluer le paramètre `testMe` de la procédure `checkIt` et retourne soit le mot "Large" si le montant est supérieur à 1 000, soit le mot "Small" s'il est inférieur à 1 000.

VB

```
Function checkIt(ByVal testMe As Integer) As String
    Return CStr(IIf(testMe > 1000, "Large", "Small"))
End Function
```

Notez que si **Option Strict** a la valeur **On**, vous devez utiliser le mot clé **CStr** pour convertir explicitement le retour de **Object** vers **String**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Option Strict, instruction
Fonctions de conversion de types de données
Choose, fonction
If...Then...Else, instruction (Visual Basic)
Select...Case, instruction (Visual Basic)
Switch, fonction

Input, fonction

Lit des données dans un fichier séquentiel ouvert et les assigne à des variables.

```
Public Sub Input( _
   FileNumber As Integer, _
   ByRef Value As Object _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Value

Requis. Variable à laquelle sont assignées les valeurs lues dans le fichier. Ne peut être une variable tableau ou objet.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Input** est fournie pour une compatibilité descendante et peut avoir un impact sur la performance. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Les données lues à l'aide de la fonction **Input** sont généralement écrites dans un fichier avec la fonction **Write**. N'utilisez cette fonction qu'avec des fichiers ouverts en mode **Input** ou **Binary**.

Remarque de sécurité
Lors de la lecture de fichiers, ne décidez pas du contenu du fichier en vous basant sur l'extension de son nom. Par exemple, u n fichier nommé Form1.vb peut ne pas être un fichier source Visual Basic 2005.

Lorsqu'elles sont lues, les chaînes standard ou les données numériques sont assignées à des variables sans être modifiées. Le tableau suivant indique comment les autres données d'entrée sont traitées :

Données	Valeur assignée à la variable
Virgule de séparation ou ligne vide	Vide
#NULL#	**DBNull**
#TRUE# ou #FALSE#	**True** ou **False**
#yyyy-mm-dd hh:mm:ss#	Date et/ou heure représentées par l'expression
#ERROR *errornumber*#	*errornumber* (la variable est un objet avec les mêmes balises qu'une erreur)

Si la fin du fichier est atteinte pendant l'entrée d'un élément de données, cela met fin à l'entrée et une erreur se produit.

Remarque

La fonction **Input** n'est pas localisée. Donc, dans la version allemande, si vous entrez 3,14159, seule la valeur 3 sera retourné e puisque la virgule est considérée comme un séparateur de variable et non comme une virgule décimale.

Remarque de sécurité

La lecture d'un fichier à l'aide de la fonction **Input** requiert l'accès **Read** à partir de l'énumération **FileIOPermissionAccess**. Pour plus d'informations, consultez FileIOPermissionAccess, énumération.

Exemple

Cet exemple utilise la fonction **Input** pour lire les données d'un fichier et les placer dans deux variables. Cet exemple suppose que le fichier `TestFile` contient quelques lignes de données écrites à l'aide de la fonction **Write**, chaque ligne contenant une chaîne entre guillemets et un nombre, séparés par une virgule, par exemple ("Hello", 234).

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output)
Write(1, "hello")
Write(1, 14)
FileClose(1)
Dim s As String = "teststring"
Dim i As Integer
FileOpen(1, "TESTFILE", OpenMode.Input)
Input(1, s)
MsgBox(s)
Input(1, i)
MsgBox(i)
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : insérer du texte dans des fichiers dans Visual Basic

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Référence

InputString, fonction

FileOpen, fonction

Fonctions Print, PrintLine

Fonctions Write, WriteLine

Autres ressources

Accès au fichier avec Visual Basic

InputBox, fonction (Visual Basic)

Affiche une invite dans une boîte de dialogue, attend que l'utilisateur tape du texte ou clique sur un bouton, puis retourne le contenu de la zone de texte sous la forme d'une chaîne.

```
Public Function InputBox( _
   ByVal Prompt As String, _
   Optional ByVal Title As String = "", _
   Optional ByVal DefaultResponse As String = "", _
   Optional ByVal Xpos As Integer = -1, _
   Optional ByVal YPos As Integer = -1 _
) As String
```

Paramètres

Prompt

Expression **String** requise affichée comme message de la boîte de dialogue. La longueur maximale de l'argument *Prompt* est d'environ 1 024 caractères selon la largeur des caractères utilisés. Si *Prompt* se compose de plusieurs lignes, vous pouvez séparer les lignes à l'aide d'un retour chariot (13 **Chr( )**), d'un saut de ligne (10 **Chr( )**) ou d'une combinaison des deux (13 **Chr( )**& 10 **Chr( )**).

Title

Facultatif. Expression **String** affichée dans la barre de titre de la boîte de dialogue. Si l'argument *Title* est omis, le nom de l'application est placé dans la barre de titre.

DefaultResponse

Facultatif. Expression **String** affichée par défaut dans la zone de texte en l'absence de toute autre valeur. Si *DefaultResponse* est omis, la zone de texte qui s'affiche est vide.

XPos

Facultatif. Expression numérique qui spécifie, en pixels, la distance entre le bord gauche de la boîte de dialogue et le bord gauche de l'écran. Si les arguments *XPos* et *YPos* sont omis, la boîte de dialogue est centrée par rapport à l'écran.

YPos

Facultatif. Expression numérique qui spécifie, en pixels, la distance entre le bord supérieur de la boîte de dialogue et le haut de l'écran. Si les arguments *XPos* et *YPos* sont omis, la boîte de dialogue est centrée par rapport à l'écran.

Notes

Si l'utilisateur clique sur **Annuler**, une chaîne de longueur nulle est retournée.

Si vous souhaitez préciser plus d'informations que celles fournies par le premier argument, vous devez utiliser la fonction **InputBox** dans une expression. Si vous omettez certains arguments de position, vous devez quand même conserver la virgule de séparation correspondante.

Remarque

La fonction **InputBox** nécessite **UIPermission** au niveau SafeTopLevelWindows, ce qui peut affecter son exécution dans les situations où le niveau de confiance n'est pas total. Pour plus d'informations, consultez Demande d'autorisations et UIPermission, classe.

Exemple

Cet exemple montre comment utiliser la fonction **InputBox** pour demander à l'utilisateur d'entrer une valeur. Si vous omettez de spécifier les positions de x et de y, les deux axes correspondants sont automatiquement déterminés de manière à centrer la boîte de dialogue. La variable `MyValue` contient la valeur entrée par l'utilisateur si celui-ci clique sur OK ou appuie sur la touche ENTRÉE.

VB

```
Dim message, title, defaultValue As String
Dim myValue As Object
```

```
' Set prompt.
message = "Enter a value between 1 and 3"
' Set title.
title = "InputBox Demo"
defaultValue = "1"    ' Set default value.

' Display message, title, and default value.
myValue = InputBox(message, title, defaultValue)
' If user has clicked Cancel, set myValue to defaultValue
If myValue Is "" Then myValue = defaultValue

' Display dialog box at position 100, 100.
myValue = InputBox(message, title, defaultValue, 100, 100)
' If user has clicked Cancel, set myValue to defaultValue
If myValue Is "" Then myValue = defaultValue
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

MsgBox, fonction (Visual Basic)

Fonctions Chr, ChrW

InputString, fonction

Retourne une valeur **String** contenant les caractères d'un fichier ouvert en mode **Input** ou **Binary**.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **InputString**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
InputString(_
   ByVal FileNumber As Integer, _
   ByVal CharCount As Integer _
) As String
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

CharCount

Requis. Toute expression numérique valide spécifiant le nombre de caractères à lire.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
ArgumentException	5	*CharCount* < 0 ou > 214.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **InputString** est fournie pour une compatibilité descendante et peut avoir un impact sur la performance. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Les données lues à l'aide de la fonction **InputString** sont généralement écrites dans un fichier à l'aide de la fonction **Print** ou **FilePut**. N'utilisez cette fonction qu'avec des fichiers ouverts en mode **Input** ou **Binary**.

Contrairement à la fonction **Input**, la fonction **InputString** retourne tous les caractères lus, y compris les virgules, les retours chariot, les sauts de ligne, les guillemets et les espaces à gauche.

Dans le cas de fichiers ouverts en mode **Binary**, une erreur se produit si vous tentez de lire le fichier à l'aide de la fonction **InputString** jusqu'à ce que la fonction **EOF** retourne la valeur **True**. Utilisez les fonctions **LOF** et **Loc** au lieu de **EOF** pour la lecture de fichiers binaires à l'aide de **InputString** ou utilisez **FileGet** avec la fonction **EOF**.

Remarque de sécurité
Lors de la lecture de fichiers, ne décidez pas du contenu du fichier en vous basant sur l'extension de son nom. Par exemple, u n fichier nommé Form1.vb peut ne pas être un fichier source Visual Basic.

Exemple

Cet exemple utilise la fonction **InputString** pour lire un fichier caractère par caractère et l'afficher dans la fenêtre **Output**. Cet exemple suppose que `MyFile` est un fichier texte comportant quelques lignes d'exemples de données.

VB

```
Dim oneChar As String
' Open file.
FileOpen(1,  "MYFILE.TXT", OpenMode.Input)
```

```
' Loop until end of file.
While Not EOF(1)
' Get one character.
oneChar = (InputString(1, 1))
' Print to the output window.
System.Console.Out.WriteLine(oneChar)
End While
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Comment : insérer du texte dans des fichiers dans Visual Basic

Référence

Input, fonction

Autres ressources

Accès au fichier avec Visual Basic

InStr, fonction (Visual Basic)

Retourne un entier spécifiant la position de début de la première occurrence d'une chaîne à l'intérieur d'une autre.

```
Public Shared Function InStr(_
   ByVal String1 As String, _
   ByVal String2 As String, _
   Optional ByVal Compare As CompareMethod _
) As Integer
' -or-
Public Shared Function InStr(_
   ByVal Start As Integer, _
   ByVal String1 As String, _
   ByVal String2 As String, _
   Optional ByVal Compare As Microsoft.VisualBasic.CompareMethod _
) As Integer
```

Paramètres

Start

Facultatif. Expression numérique qui définit la position de départ de chaque recherche. Si cet argument est omis, la recherche commence à la position du premier caractère. L'index de début est de base 1.

String1

Requis. Expression **String** recherchée.

String2

Requis. Expression **String** recherchée.

Compare

Facultatif. Spécifie le type de comparaison de chaînes. Si *Compare* est omis, le paramètre **Option Compare** détermine le type de comparaison.

Paramètres

L'argument *Compare* peut prendre les valeurs suivantes :

Constante	Valeur	Description
Binary	0	Effectue une comparaison binaire.
Text	1	Effectue une comparaison textuelle.

Valeur de retour

Si	La fonction InStr retourne
String1 est de longueur nulle ou **Nothing**	0
String2 est de longueur nulle ou **Nothing**	*start*
String2 est introuvable	0
String2 est trouvé dans *String1*	Position où commence la correspondance
Start > *String2*	0

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Start* < 1.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

En général, la fonction **InStr** est utilisée lors de l'analyse de chaînes.

Remarque

Dans les précédentes versions de Visual Basic, la fonction **InStrB** retourne un nombre d'octets plutôt qu'une position de caractère. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Character Set). Toutes les chaînes Visual Basic 2005 sont en Unicode et la fonction **InStrB** n'est plus prise en charge.

Exemple

L'exemple suivant utilise la fonction **InStr** pour retourner la position de la première occurrence d'une chaîne à l'intérieur d'une autre.

VB

```
' String to search in.
Dim SearchString As String = "XXpXXpXXPXXP"
' Search for "P".
Dim SearchChar As String = "P"

Dim TestPos As Integer
' A textual comparison starting at position 4. Returns 6.
TestPos = InStr(4, SearchString, SearchChar, CompareMethod.Text)

' A binary comparison starting at position 1. Returns 9.
TestPos = InStr(1, SearchString, SearchChar, CompareMethod.Binary)

' If Option Compare is not set, or set to Binary, return 9.
' If Option Compare is set to Text, returns 3.
TestPos = InStr(SearchString, SearchChar)

' Returns 0.
TestPos = InStr(1, SearchString, "W")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

InStrRev, fonction (Visual Basic)
Option Compare, instruction
StrComp, fonction (Visual Basic)
ArgumentException

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Autres ressources

Chaînes en Visual Basic
Introduction aux chaînes dans Visual Basic

InStrRev, fonction (Visual Basic)

Retourne la position de la première occurrence d'une chaîne dans une autre, à partir du côté droit de la chaîne.

```
Public Function InStrRev(
   ByVal StringCheck As String,
   ByVal StringMatch As String,
   Optional ByVal Start As Integer = -1,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
) As Integer
```

Paramètres

StringCheck

Requis. Expression de chaîne à parcourir.

StringMatch

Requis. Expression de chaîne à rechercher.

Start

Facultatif. Expression numérique qui définit la position de départ (base 1) de chaque recherche, à partir du côté gauche de la chaîne. Si *Start* est omis, la valeur -1 est employée, ce qui signifie que la recherche commence à la position du dernier caractère. La recherche se poursuit de droite à gauche.

Compare

Facultatif. Valeur numérique indiquant le type de comparaison à utiliser pour l'évaluation des sous-chaînes. En cas d'omission, une comparaison binaire est effectuée. Consultez la section Valeurs pour obtenir les valeurs.

Paramètres

L'argument *Compare* peut prendre les valeurs suivantes.

Constante	Description
Binary	Effectue une comparaison binaire
Text	Effectue une comparaison de texte

Valeur de retour

InStrRev retourne les valeurs suivantes.

Si	Valeur retournée par la fonction InStrRev
StringCheck est de longueur nulle	0
StringMatch est de longueur nulle	*Start*
StringMatch est introuvable	0
StringMatch est trouvé dans *StringCheck*	Position à laquelle la première correspondance est trouvée, à partir du côté droit de la chaîne.
Start est supérieur à la longueur de *StringMatch*	0

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Start* = 0 ou *Start* < -1.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La syntaxe de la fonction **InStrRev** est différente de celle de la fonction **InStr**.

Exemple

Cet exemple illustre l'utilisation de la fonction **InStrRev**.

VB

```
Dim TestString As String = "the quick brown fox jumps over the lazy dog"
Dim TestNumber As Integer
' Returns 32.
TestNumber = InStrRev(TestString, "the")
' Returns 1.
TestNumber = InStrRev(TestString, "the", 16)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

InStr, fonction (Visual Basic)

Autres ressources

Chaînes en Visual Basic

Introduction aux chaînes dans Visual Basic

Fonctions Int, Fix (Visual Basic)

Retournent la partie entière d'un nombre.

```
Public Shared Function Int( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
Public Shared Function Fix( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
```

Paramètres

Number

Requis. Nombre de type **Double** ou toute expression numérique valide. Si l'argument *Number* contient une valeur de type **Nothing**, **Nothing** est retourné.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	5	Number n'est pas spécifié.
ArgumentException	5	Number n'est pas un type numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les fonctions **Int** et **Fix** suppriment la partie fractionnaire de *Number* et retournent la valeur entière résultante.

La différence entre les fonctions **Int** et **Fix** réside dans le fait que si *Number* est négatif, **Int** retourne le premier entier négatif inférieur ou égal à *Number*, alors que **Fix** retourne le premier entier négatif supérieur ou égal à *Number*. Par exemple, **Int** convertit -8,4 en -9 et **Fix** convertit -8,4 en -8.

`Fix(number)` équivaut à `Sign(number) * Int(Abs(number))`.

Exemple

Cet exemple illustre de quelle façon les fonctions **Int** et **Fix** retournent la partie entière des nombres. Si l'argument est négatif, la fonction **Int** retourne le premier entier négatif inférieur ou égal au nombre ; la fonction **Fix** retourne le premier entier négatif supérieur ou égal au nombre. L'exemple suivant nécessite que vous spécifiiez **Option Strict Off**, car les conversions implicites du type **Double** en type **Integer** ne sont pas autorisées sous **Option Strict On** :

VB

```
' This code requires Option Strict Off
Dim MyNumber As Integer
MyNumber = Int(99.8)   ' Returns 99.
MyNumber = Fix(99.8)   ' Returns 99.

MyNumber = Int(-99.8)  ' Returns -100.
MyNumber = Fix(-99.8)  ' Returns -99.

MyNumber = Int(-99.2)  ' Returns -100.
MyNumber = Fix(-99.2)  ' Returns -99.
```

Vous pouvez utiliser la fonction **CInt** pour convertir explicitement d'autres types de données en type **Integer** avec **Option Strict Off**. Toutefois, **CInt** est arrondi à l'entier le plus proche au lieu de tronquer la partie fractionnaire des nombres. Par exemple :

VB

```
MyNumber = CInt(99.8)    ' Returns 100.
MyNumber = CInt(-99.8)   ' Returns -100.
MyNumber = CInt(-99.2)   ' Returns -99.
```

Vous pouvez utiliser la fonction **CInt** sur le résultat d'un appel à **Fix** ou à **Int** afin d'effectuer une conversion explicite en entier sans arrondi. Par exemple :

VB

```
MyNumber = CInt(Fix(99.8))   ' Returns 99.
MyNumber = CInt(Int(99.8))   ' Returns 99.
```

Pour plus d'informations sur **CInt**, consultez Fonctions de conversion de types de données.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Fonctions de conversion de types de données
Integer, type de données (Visual Basic)
Liste des mots clés mathématiques
Fonctions mathématiques (Visual Basic)
Liste des conversions
ArgumentNullException

IPmt, fonction

Retourne une valeur de type **Double** indiquant les intérêts, pour une période donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.

```
Function IPmt( _
   ByVal Rate As Double, _
   ByVal Per As Double, _
   ByVal NPer As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

Paramètres

Rate

Requis. **Double** spécifiant le taux d'intérêt par période. Par exemple, si vous obtenez un prêt pour l'achat d'une voiture à un taux d'intérêt annuel (TIA) de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

Per

Requis. Valeur de type **Double** indiquant l'échéance, comprise entre 1 et *NPer*.

NPer

Requis. **Double** spécifiant le nombre total d'échéances de l'annuité. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt d'une durée de quatre ans pour l'achat d'une voiture, votre emprunt comporte un total de 4 x 12 (soit 48) échéances.

PV

Requis. Donnée de type **Double** indiquant la valeur actuelle, ou valeur au jour d'aujourd'hui, d'une série de paiements ou d'encaissements futurs. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond pour le prêteur à la valeur actuelle des remboursements mensuels que vous allez effectuer pour la voiture.

FV

Facultatif. **Double** spécifiant la valeur future ou le solde de caisse souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 F, car il s'agit de sa valeur après le dernier remboursement. Toutefois, si vous souhaitez économiser 50 000 F sur 18 ans pour les études de votre enfant, ce montant constitue la valeur future. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être `DueDate.EndOfPeriod` si les paiements sont dus à terme échu ou `DueDate.BegOfPeriod` si les paiements sont dus à terme à échoir. Si cet argument est omis, `DueDate.EndOfPeriod` est utilisé par défaut.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Per* <= 0 ou *Per* > *NPer*

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Les arguments *Rate* et *NPer* doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités. Par exemple, si *Rate* est exprimé en mois, il doit en être de même pour *NPer*.

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Exemple

Cet exemple utilise la fonction **IPmt** pour calculer la proportion des intérêts dans le montant des remboursements d'un emprunt sachant que le montant des remboursements reste constant. Sont spécifiés le taux d'intérêt par période (`APR / 12`), la période de remboursement pour laquelle on souhaite connaître la proportion des intérêts (`Period`), le nombre total des remboursements (`TotPmts`), la valeur actuelle ou le capital de l'emprunt (`PVal`), la valeur future de l'emprunt (`FVal`) et un nombre indiquant si les remboursements s'effectuent au début ou à la fin des échéances (`PayType`).

VB

```
Sub TestIPMT()
    Dim APR, PVal, Period, IntPmt, TotInt, TotPmts As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Usually 0 for a loan.
    Dim Fval As Double = 0
    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    If APR > 1 Then APR = APR / 100 ' Ensure proper form.
    TotPmts = CInt(InputBox("How many monthly payments?"))
    Response = MsgBox("Do you make payments at end of the month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    For Period = 1 To TotPmts   ' Total all interest.
        IntPmt = IPmt(APR / 12, Period, TotPmts, -PVal, Fval, PayType)
        TotInt = TotInt + IntPmt
    Next Period

    ' Display results.
    MsgBox("You will pay a total of " & Format(TotInt, Fmt) & _
        " in interest for this loan.")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

ArgumentException

IRR, fonction

Retourne une donnée de type **Double** indiquant le taux de rendement interne d'une série de liquidités périodiques (paiements et encaissements).

```
Function IRR( _
    ByRef ValueArray() As Double, _
   Optional ByVal Guess As Double = 0.1 _
) As Double
```

Paramètres

ValueArray

Requis. Tableau de valeurs **Double** représentant les valeurs des liquidités. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement).

Guess

Facultatif. Objet indiquant la valeur qui devrait être retournée par **IRR**. Si cet argument est omis, *Guess* a la valeur 0.1 (10 pour cent).

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Les valeurs de l'argument tableau ne sont pas valides ou *Guess* <= -1.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Le taux de rendement interne est le taux d'intérêt perçu pour un investissement comprenant des paiements et des encaissements survenant à intervalles réguliers.

La fonction **IRR** utilise l'ordre des valeurs dans le tableau pour interpréter l'ordre des paiements et des encaissements. Veillez à entrer dans le bon ordre les valeurs représentant vos paiements et vos encaissements. Les liquidités de chaque période ne doivent pas nécessairement être fixes, comme dans le cas d'une annuité.

IRR est calculé par itération. En commençant par la valeur de l'argument *Guess*, la fonction **IRR** répète le cycle de calcul jusqu'à ce que le résultat soit précis à 0,00001 pour cent près. La fonction **IRR** échoue si elle n'a pas pu trouver de résultat après 20 tentatives.

Exemple

Dans cet exemple, la fonction **IRR** retourne le taux de rendement interne d'une série de cinq liquidités contenues dans le tableau `Values()`. Le premier élément du tableau est un mouvement de trésorerie négatif correspondant aux frais de lancement de l'entreprise. Les quatre liquidités restantes représentent des liquidités positives pour les quatre années suivantes. `Guess` est le taux de rendement interne estimé.

VB

```
' Define money format.
Dim MoneyFmt As String = "###,##0.00"
' Define percentage format.
Dim PercentFmt As String = "#0.00"

Dim values(4) As Double
' Business start-up costs.
values(0) = -70000
' Positive cash flows reflecting income for four successive years.
values(1) = 22000
values(2) = 25000
values(3) = 28000
```

```
values(4) = 31000

' Use the IRR function to calculate the rate of return.
' Guess starts at 10 percent.
Dim Guess As Double = 0.1
' Calculate internal rate.
Dim CalcRetRate As Double = IRR(values, Guess) * 100
' Display internal return rate.
MsgBox("The internal rate of return for these cash flows is " & _
    Format(CalcRetRate, CStr(PercentFmt)) & " percent.")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

NPV, fonction
MIRR, fonction
Liste des mots clés financiers
ArgumentException

IsArray, fonction (Visual Basic)

Retourne une valeur **Boolean** indiquant si une variable pointe vers un tableau.

```
Public Function IsArray(ByVal VarName As Object) As Boolean
```

Paramètres
VarName

Requis. Variable **Object**.

Notes

IsArray retourne **True** si la variable pointe vers un tableau ; sinon, retourne **False**. **IsArray** est particulièrement utile avec les objets qui peuvent contenir des tableaux.

Exemple

Cet exemple utilise la fonction **IsArray** pour vérifier si plusieurs variables font référence à un tableau.

VB

```
Dim firstArray(4), secondArray(3) As Integer
Dim thisString As String = "Test"
Dim arrayCheck As Boolean
arrayCheck = IsArray(firstArray)
arrayCheck = IsArray(secondArray)
arrayCheck = IsArray(thisString)
' The first two calls to IsArray return True; the third returns False.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches
Comment : déclarer une variable tableau

Référence
IsDate, fonction (Visual Basic)
IsDBNull, fonction
IsError, fonction
IsNothing, fonction
IsNumeric, fonction (Visual Basic)
IsReference, fonction
Object, type de données
TypeName, fonction (Visual Basic)

IsDate, fonction (Visual Basic)

Retourne une valeur **Boolean** indiquant si une expression représente une valeur **Date** valide.

```
Public Function IsDate(ByVal Expression As Object) As Boolean
```

Paramètres
Expression

Requis. Expression **Object**.

Notes

IsDate retourne **True** si *Expression* est de type de données Date ou peut être converti en **Date** ; sinon, il retourne **False**.

Le type de données **Date** maintient à la fois une valeur de date et une valeur d'heure. **IsDate** retourne **True** si *Expression* représente une date valide, une heure valide ou une date et heure valides.

Exemple

L'exemple suivant utilise la fonction **IsDate** pour déterminer si plusieurs variables représentent des valeurs **Date** valides.

VB

```
Dim firstDate, secondDate As Date
Dim timeOnly, dateAndTime, noDate As String
Dim dateCheck As Boolean
firstDate = CDate("February 12, 1969")
secondDate = #2/12/1969#
timeOnly = "3:45 PM"
dateAndTime = "March 15, 1981 10:22 AM"
noDate = "Hello"
dateCheck = IsDate(firstDate)
dateCheck = IsDate(secondDate)
dateCheck = IsDate(timeOnly)
dateCheck = IsDate(dateAndTime)
dateCheck = IsDate(noDate)
```

Dans l'exemple précédent, **IsDate** retourne **True** pour les quatre premiers appels et **False** pour le dernier appel.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
IsArray, fonction (Visual Basic)
IsDBNull, fonction
IsError, fonction
IsNothing, fonction
IsNumeric, fonction (Visual Basic)
IsReference, fonction
Object, type de données
Type de données date (Visual Basic)
TypeName, fonction (Visual Basic)

IsDBNull, fonction

Retourne une valeur **Boolean** indiquant si une expression correspond à la classe System.DBNull.

```
Public Function IsDBNull(ByVal Expression As Object) As Boolean
```

Paramètres
Expression

Requis. Expression **Object**.

Notes

IsDBNull retourne **True** si le type de données de *Expression* correspond au type **DBNull** ; sinon, **IsDBNull** retourne **False**.

La valeur **System.DBNull** indique que le **Object** représente des données manquantes ou inexistantes. **DBNull** est différent de **Nothing** qui indique qu'une variable n'a pas encore été initialisée. **DBNull** est également différent d'une chaîne de longueur nulle (**""**) qu'on appelle aussi parfois chaîne vide.

Exemple

Cet exemple utilise la fonction **IsDBNull** pour déterminer si une variable correspond à **DBNull**.

VB

```
Dim testVar As Object
Dim nullCheck As Boolean
nullCheck = IsDBNull(testVar)
testVar = ""
nullCheck = IsDBNull(testVar)
testVar = System.DBNull.Value
nullCheck = IsDBNull(testVar)
' The first two calls to IsDBNull return False; the third returns True.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
IsArray, fonction (Visual Basic)
IsDate, fonction (Visual Basic)
IsError, fonction
IsNothing, fonction
IsNumeric, fonction (Visual Basic)
IsReference, fonction
Object, type de données
TypeName, fonction (Visual Basic)
DBNull

IsError, fonction

Retourne une valeur **Boolean** indiquant si une expression est un type d'exception.

```
Public Function IsError(ByVal Expression As Object) As Boolean
```

Paramètres
Expression

Requis. Expression **Object**.

Notes

IsError retourne **True** si l'expression représente une variable **Object** qui dérive de la classe Exception dans l'espace de noms System.

Une exception qui dérive de **System.Exception** peut être interceptée avec les instructions **Try...Catch...Finally**.

Exemple

L'exemple suivant utilise la fonction **IsError** pour vérifier si une expression représente une exception système.

VB

```
Sub demonstrateIsError(ByVal firstArg As Integer)
    Dim returnVal As New Object
    Dim badArg As String = "Bad argument value"
    Dim errorCheck As Boolean
    If firstArg > 10000 Then
        returnVal = New System.ArgumentOutOfRangeException(badArg)
    End If
    errorCheck = IsError(returnVal)
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
IsArray, fonction (Visual Basic)
IsDate, fonction (Visual Basic)
IsDBNull, fonction
IsNothing, fonction
IsNumeric, fonction (Visual Basic)
IsReference, fonction
Object, type de données
TypeName, fonction (Visual Basic)

IsNothing, fonction

Retourne une valeur **Boolean** indiquant si aucun objet n'est assigné à une expression.

```
Public Function IsNothing(ByVal Expression As Object) As Boolean
```

Paramètres
Expression

Requis. Expression **Object**.

Notes

IsNothing retourne **True** si l'expression représente une variable objet à laquelle aucun objet n'est actuellement assigné ; sinon, il retourne **False**.

IsNothing est prévu pour utiliser les types référence. Un type valeur ne peut pas contenir de valeur de Nothing et rétablit sa valeur par défaut si vous lui assignez **Nothing**. Si vous fournissez un type valeur dans *Expression*, **IsNothing** retourne toujours **False**.

Exemple

L'exemple suivant utilise la fonction **IsNothing** pour déterminer si une variable objet est associée à une instance d'objet.

VB

```
Dim testVar As Object
' No instance has been assigned to variable testVar yet.
Dim testCheck As Boolean
' The following call returns True.
testCheck = IsNothing(testVar)
' Assign a string instance to variable testVar.
testVar = "ABCDEF"
' The following call returns False.
testCheck = IsNothing(testVar)
' Disassociate variable testVar from any instance.
testVar = Nothing
' The following call returns True.
testCheck = IsNothing(testVar)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
IsArray, fonction (Visual Basic)
IsDate, fonction (Visual Basic)
IsDBNull, fonction
IsError, fonction
IsNumeric, fonction (Visual Basic)
IsReference, fonction
Object, type de données
TypeName, fonction (Visual Basic)
Concepts
Types valeur et types référence

IsNumeric, fonction (Visual Basic)

Retourne une valeur **Boolean** indiquant si une expression peut être évaluée comme un nombre.

```
Public Function IsNumeric(ByVal Expression As Object) As Boolean
```

Paramètres
Expression

Requis. Expression **Object**.

Notes

IsNumeric retourne **True** si le type de données de *Expression* est **Boolean**, **Byte**, **Decimal**, **Double**, **Integer**, **Long**, **SByte**, **Short**, **Single**, **UInteger**, **ULong** ou **UShort**, ou un **Object** contenant l'un de ces types numériques. Il retourne également **True** si *Expression* est un **Char** ou **String** qui peut être correctement converti en nombre.

IsNumeric retourne **False** si *Expression* a pour type de données **Date** ou **Object** et qu'il ne contient pas de type numérique. **IsNumeric** retourne **False** si *Expression* est un **Char** ou **String** qui ne peut pas être converti en un nombre.

Exemple

L'exemple suivant utilise la fonction **IsNumeric** pour déterminer si le contenu d'une variable peut être évalué comme un nombre.

VB

```
Dim testVar As Object
Dim numericCheck As Boolean
testVar = "53"
' The following call to IsNumeric returns True.
numericCheck = IsNumeric(testVar)
testVar = "459.95"
' The following call to IsNumeric returns True.
numericCheck = IsNumeric(testVar)
testVar = "45 Help"
' The following call to IsNumeric returns False.
numericCheck = IsNumeric(testVar)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
IsArray, fonction (Visual Basic)
IsDate, fonction (Visual Basic)
IsDBNull, fonction
IsError, fonction
IsNothing, fonction
IsReference, fonction
Object, type de données
TypeName, fonction (Visual Basic)

IsReference, fonction

Retourne une valeur **Boolean** indiquant si une expression correspond à un type référence.

```
Public Function IsReference(ByVal Expression As Object) As Boolean
```

Paramètres
Expression

Requis. Expression **Object**.

Notes

IsReference retourne **True** si *Expression* représente un type référence, tel qu'une instance de classe, un type **String** ou un tableau d'un type quelconque ; sinon, elle retourne **False**.

Un type référence contient un pointeur vers des données stockées à un autre endroit de la mémoire. Un type valeur contient ses propres données.

Exemple

Cet exemple utilise la fonction **IsReference** pour vérifier si plusieurs variables font référence à des types référence.

VB

```
Dim testArray(3) As Boolean
Dim testString As String = "Test string"
Dim testObject As Object = New Object()
Dim testNumber As Integer = 12
testArray(0) = IsReference(testArray)
testArray(1) = IsReference(testString)
testArray(2) = IsReference(testObject)
testArray(3) = IsReference(testNumber)
```

Dans l'exemple précédent, les trois premiers appels à **IsReference** retournent **True**. Le dernier appel retourne **False**, parce que **Integer** est un type valeur, et non un type référence.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
IsArray, fonction (Visual Basic)
IsDate, fonction (Visual Basic)
IsDBNull, fonction
IsError, fonction
IsNothing, fonction
IsNumeric, fonction (Visual Basic)
Object, type de données
TypeName, fonction (Visual Basic)
Concepts
Types valeur et types référence

Join, fonction (Visual Basic)

Retourne une chaîne créée par la jonction de plusieurs sous-chaînes contenues dans un tableau.

```
Function Join(
   ByVal SourceArray() As { Object | String },
   Optional ByVal Delimiter As String = " "
) As String
```

Paramètres

SourceArray

Requis. Tableau à une dimension contenant les sous-chaînes à joindre.

Delimiter

Facultatif. Toute chaîne, utilisée pour séparer les sous-chaînes dans la chaîne retournée. Si cet argument est omis, le caractère espace (" ") est employé. Si *Delimiter* est une chaîne de longueur nulle ("") ou **Nothing**, tous les éléments de la liste sont concaténés sans délimiteur.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*SourceArray* n'est pas unidimensionnel.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une parité existe entre les fonctions **Join** et **Split**. La fonction **Join** prend un tableau de chaînes et les joint à l'aide d'une chaîne de délimiteur, pour retourner une chaîne unique. La fonction **Split** prend une chaîne et la sépare au niveau du délimiteur, pour retourner un tableau de chaînes. Toutefois, une différence essentielle est que **Join** peut concaténer des chaînes avec toute chaîne de délimiteur, alors que **Split** peut uniquement séparer des chaînes à l'aide d'un délimiteur de caractère unique.

Exemple

L'exemple suivant montre comment utiliser la fonction **Join** pour créer une liste à partir de plusieurs chaînes.

VB

```
Dim TestItem() As String = {"Pickle", "Pineapple", "Papaya"}
' Returns "Pickle, Pineapple, Papaya"
Dim TestShoppingList As String = Join(TestItem, ", ")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Split, fonction (Visual Basic)
ArgumentException

Kill, fonction

Supprime des fichiers d'un disque.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **Kill**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub Kill(ByVal PathName As String)
```

Paramètres
PathName

Requis. Expression **String** qui spécifie un ou plusieurs noms de fichiers à supprimer. *PathName* peut inclure le répertoire ou le dossier et le lecteur.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	55	Le ou les fichiers cibles sont ouverts.
FileNotFoundException	53	Le ou les fichiers cibles sont introuvables.
SecurityException	Autorisation refusée (Visual Basic)	autorisation refusée ;

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Kill** prend en charge l'utilisation de caractères génériques à plusieurs caractères (*****) et à un seul caractère (**?**) pour spécifier plusieurs fichiers.

Remarque de sécurité Pour pouvoir être exécutée, la fonction **Kill** suppose que les indicateurs **Read** et **PathDiscovery** de FileIOPermission soient accordés au code en cours d'exécution. Pour plus d'informations, consultez **SecurityException** et Code Access Permissions.

Exemple

Cet exemple utilise la fonction **Kill** pour supprimer un fichier d'un disque.

VB

```
' Assume TESTFILE is a file containing some data.
Kill("TestFile")   ' Delete file.

' Delete all *.TXT files in current directory.
Kill("*.TXT")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence

RmDir, fonction
IOException Class
FileNotFoundException Class

LBound, fonction (Visual Basic)

Retourne le plus petit indice disponible pour la dimension indiquée d'un tableau.

```
Public Function LBound( _
   ByVal Array As System.Array, _
   Optional ByVal Rank As Integer = 1 _
) As Integer
```

Paramètres

Array

Requis. Tableau de tout type de données. Tableau dans lequel vous voulez trouver le plus petit indice possible d'une dimension.

Rank

Facultatif. **Integer**. Dimension pour laquelle le plus petit indice possible doit être retourné. Utilisez la valeur 1 pour la première dimension, la valeur 2 pour la deuxième, etc. Si l'argument *Rank* est omis, 1 est supposé.

Valeur de retour

Integer. La valeur la plus faible que l'indice pour la dimension spécifiée peut contenir. **LBound** retourne toujours 0 à condition que le *Array* ait été initialisé, même s'il n'a pas d'éléments, par exemple si c'est une chaîne de longueur nulle. Si le *Array* est **Nothing**, **LBound** lève un ArgumentNullException.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	9	Le *Array* a la valeur **Nothing**.
RankException	9	*Rank* est inférieur à 1 ou *Rank* supérieur au rang de *Array*.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Compte tenu que les indices de tableau commencent à 0, le plus petit indice disponible pour chaque dimension est toujours 0.

Pour un tableau avec les dimensions suivantes, **LBound** retourne les valeurs dans le tableau suivant :

```
Dim a(100, 5, 4) As Byte
```

Appel à LBound	Valeur de retour
`LBound(a, 1)`	0
`LBound(a, 2)`	0
`LBound(a, 3)`	0

Exemple

Cet exemple utilise la fonction **LBound** pour déterminer le plus petit indice disponible pour la dimension indiquée d'un tableau.

VB

```
Dim lowest, bigArray(10, 15, 20), littleArray(6) As Integer
lowest = LBound(bigArray, 1)
```

```
lowest = LBound(bigArray, 3)
lowest = LBound(littleArray)
' All three calls to LBound return 0.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

UBound, fonction (Visual Basic)
Dim, instruction (Visual Basic)
ReDim, instruction (Visual Basic)
ArgumentException
RankException

LCase, fonction (Visual Basic)

Retourne une chaîne ou un caractère converti en lettres minuscules.

```
Public Shared Function LCase(ByVal Value As Char) As Char
' -or-
Public Shared Function LCase(ByVal Value As String) As String
```

Paramètres
Value

Requis. Toute expression **String** ou **Char** valide.

Notes

Seules les majuscules sont converties en minuscules ; toutes les lettres en minuscules et les caractères autres que les lettres demeurent inchangés.

Cette fonction utilise les informations relatives à la culture de l'application lors de la manipulation de la chaîne afin que les changements de casse soient appropriés pour les paramètres régionaux dans lesquels l'application est utilisée.

Remarque de sécurité

Si votre application prend des décisions relatives à la sécurité en fonction du résultat d'une comparaison ou d'une opération de modification des majuscules et des minuscules, l'opération doit utiliser la méthode System.String.Compare(System.String,System.String,System.StringComparison) et passer Ordinal ou OrdinalIgnoreCase pour l'argument *comparisonType*. Pour plus d'informations, consultez Comment les informations de culture affectent les chaînes dans Visual Basic.

Exemple

L'exemple suivant utilise la fonction **LCase** pour retourner une version en minuscules d'un nombre.

VB

```
' String to convert.
Dim UpperCase As String = "Hello World 1234"
' Returns "hello world 1234".
Dim LowerCase As String = LCase(UpperCase)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
UCase, fonction (Visual Basic)
Autres ressources
Chaînes en Visual Basic
Introduction aux chaînes dans Visual Basic

Left, fonction (Visual Basic)

Retourne une chaîne contenant un nombre spécifié de caractères depuis la partie gauche d'une chaîne.

```
Public Shared Function Left( _
   ByVal str As String, _
   ByVal Length As Integer _
) As String
```

Paramètres

str

Requis. Expression **String** à partir de laquelle les caractères situés à l'extrême gauche sont retournés.

Length

Requis. Expression **Integer**. Expression numérique indiquant le nombre de caractères à retourner. Pour la valeur 0, une chaîne de longueur nulle ("") est retournée. Si le nombre de caractères est supérieur ou égal à celui de *str*, la chaîne entière est retournée.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Length* < 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Pour déterminer le nombre de caractères contenus dans *str*, utilisez la fonction **Len**. Si elle est utilisée dans une application Windows Forms, ou dans toute autre classe qui a une propriété **Left**, vous devez attribuer un nom complet à la fonction avec `Microsoft.VisualBasic.Left`.

Remarque

Dans les précédentes versions de Visual Basic, la fonction **LeftB** retourne une chaîne en octets plutôt que des caractères. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Chara cter Set). Toutes les chaînes Visual Basic en cours sont en Unicode et la fonction **LeftB** n'est plus prise en charge.

Exemple

L'exemple suivant montre l'utilisation de la fonction **Left** pour retourner une sous-chaîne d'une expression **String** donnée. Dans une classe dotée d'une propriété **Left**, il peut être nécessaire d'attribuer un nom complet à la fonction **Left**.

VB

```
Dim TestString As String = "Hello World!"
' Returns "Hello".
Dim subString As String = Microsoft.VisualBasic.Left(TestString, 5)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Right, fonction (Visual Basic)
Len, fonction (Visual Basic)
Mid, fonction (Visual Basic)
ArgumentException
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Len, fonction (Visual Basic)

Retourne un entier contenant le nombre de caractères contenus dans une chaîne ou le nombre nominal d'octets requis pour stocker une variable.

```
Public Shared Function Len( _
   ByVal Expression As { Boolean | Byte | SByte | Char | Double |
   Integer | UInteger | Long | ULong | Object | Short | UShort |
   Single | String | DateTime | Decimal } _
) As Integer
```

Paramètres

Expression

Toute expression **String** ou nom de variable valide. Si *Expression* est de type **Object**, la fonction **Len** retourne la taille telle qu'elle sera écrite dans le fichier par la fonction **FilePut**.

Notes

Avec les types définis par l'utilisateur et les variables **Object**, la fonction **Len** retourne la taille telle qu'elle sera écrite dans le fichier par la fonction **FilePut**. Si **Object** contient une **String**, il retourne la longueur de la chaîne. Si **Object** contient un autre type, il retournera la taille de l'objet lors de son écriture dans le fichier par la fonction **FilePut**.

L'attribut **VBFixedString** peut être appliqué aux champs de type chaîne dans les objets pour indiquer la taille de la chaîne, en octets, lors de l'écriture sur le disque. La fonction **Len** utilise l'attribut **VBFixedString**, s'il est disponible, pour déterminer la taille des variables **Object**.

Remarque

La fonction **Len** ne peut peut-être pas déterminer le nombre réel d'octets de stockage requis si elle est utilisée avec des chaînes de longueur variable dans des types de données définis par l'utilisateur.

Remarque

Dans les précédentes versions de Visual Basic, la fonction **LenB** retourne le nombre d'octets dans une chaîne plutôt que des caractères. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Character Set). Toutes les chaînes Visual Basic en cours sont en Unicode et la fonction **LenB** n'est plus prise en charge.

Exemple

L'exemple suivant utilise la fonction **Len** pour retourner le nombre de caractères d'une chaîne.

VB

```
' Initializes variable.
Dim TestString As String = "Hello World"
' Returns 11.
Dim TestLen As Integer = Len(TestString)
```

Notes du développeur sur Smart Device

La fonction **Len** n'accepte que des variables **String** et **Object** comme paramètres. Si un **Object** contient une variable **String**, il retournera la longueur de la chaîne. Si un paramètre est une référence **Object** NULL, la fonction retourne zéro. Si un **Object** contient un autre type, une exception est levée.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des types de données (Visual Basic)

FilePut, fonction

VBFixedStringAttribute, classe

Liste des manipulations de chaînes

Liste des types de données (Visual Basic)

InStr, fonction (Visual Basic)

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

LineInput, fonction

Lit une ligne unique à partir d'un fichier séquentiel ouvert et l'assigne à une variable **String**.

```
Public Function LineInput(ByVal FileNumber As Integer) As String
```

Paramètres
FileNumber

Requis. Tout numéro de fichier valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
EndOfStreamException	62	La fin du fichier est atteinte.
IOException	52	*FileNumber* n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **LineInput** est fournie pour une compatibilité descendante et peut avoir un impact sur la performance. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Les données lues à l'aide de la fonction **LineInput** sont généralement écrites dans un fichier avec la fonction **Print**.

Remarque de sécurité
Lors de la lecture de fichiers, ne décidez pas du contenu d'un fichier en vous basant sur l'extension de son nom. Par exemple, un fichier nommé Form1.vb peut ne pas être un fichier source Visual Basic.

La fonction **LineInput** lit dans un fichier un caractère à la fois jusqu'à ce qu'elle rencontre un retour chariot (**Chr(**13**)**) ou d'une séquence saut de ligne-retour chariot (**Chr(**13**)** + **Chr(**10**)**). Les séquences retour chariot-saut de ligne ne sont pas prises en compte (elles ne sont pas ajoutées à la chaîne de caractères).

Remarque de sécurité
La lecture d'un fichier à l'aide de la fonction **LineInput** requiert l'accès **Read** à partir de l'énumération FileIOPermissionAccess.

Exemple

Cet exemple utilise la fonction **LineInput** pour lire une ligne dans un fichier séquentiel et l'assigner à une variable. Cet exemple suppose que `TestFile` est un fichier texte comportant quelques lignes d'exemples de données.

VB

```
Dim TextLine As String
' Open file.
FileOpen(1, "TESTFILE", OpenMode.Input)
' Loop until end of file.
While Not EOF(1)
   ' Read line into variable.
   TextLine = LineInput(1)
   ' Print to the console.
   WriteLine(1, TextLine)
End While
```

```
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Comment : insérer du texte dans des fichiers dans Visual Basic

Référence

Fonctions Chr, ChrW

Input, fonction

Autres ressources

Accès au fichier avec Visual Basic

Loc, fonction

Retourne une valeur **Long** spécifiant la position de lecture/écriture actuelle dans un fichier ouvert.

```
Public Function Loc(ByVal FileNumber As Integer) As Long
```

Paramètres
FileNumber

Requis. Tout numéro de fichier **Integer** valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Loc** est de base zéro; son utilisation afin d'obtenir le premier octet dans un fichier retournera 0.

La fonction **Loc** est fournie pour une compatibilité descendante et peut avoir un impact sur la performance. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Le tableau suivant décrit la valeur de retour en fonction du mode d'accès au fichier :

Mode	Valeur de retour
Random	Numéro du dernier enregistrement lu ou écrit dans le fichier.
Sequential	Position actuelle de l'octet dans le fichier divisée par 128. Cependant, les informations retournées par la fonction **Loc** pour les fichiers séquentiels ne sont toutefois ni utilisées, ni requises.
Binary	Position du dernier octet lu ou écrit.

Exemple

Cet exemple utilise la fonction **Loc** pour retourner la position de lecture/écriture actuelle dans un fichier ouvert. Cet exemple suppose que `MyFile` est un fichier texte comportant quelques lignes d'exemple de données.

VB

```
Dim location As Long
Dim oneChar As Char
FileOpen(1, "C:\TESTFILE.TXT", OpenMode.Binary)
While location < LOF(1)
   Input(1, oneChar)
   location = Loc(1)
   WriteLine(1, location & ControlChars.CrLf)
End While
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

EOF, fonction
LOF, fonction
Seek, fonction
IOException

Fonctions Lock, Unlock

Contrôle l'accès par d'autres processus à tout ou partie d'un fichier ouvert à l'aide de la fonction **Open**.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **Lock** et **Unlock**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Overloads Sub Lock(ByVal FileNumber As Integer)
' -or-
Public Overloads Sub Unlock(ByVal FileNumber As Integer)
' -or-
Public Overloads Sub Lock(_
   ByVal FileNumber As Integer, _
   ByVal Record As Long _
)
' -or-
Public Overloads Sub Unlock( _
   ByVal FileNumber As Integer, _
   ByVal Record As Long _
)
' -or-
Public Overloads Sub Lock( _
   ByVal FileNumber As Integer, _
   ByVal FromRecord As Long, _
   ByVal ToRecord As Long _
)
' -or-
Public Overloads Sub Unlock( _
   ByVal FileNumber As Integer, _
   ByVal FromRecord As Long, _
   ByVal ToRecord As Long _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Record

Facultatif. Numéro du seul enregistrement ou octet à verrouiller ou à déverrouiller

FromRecord

Facultatif. Numéro du premier enregistrement ou octet à verrouiller ou à déverrouiller.

ToRecord

Facultatif. Numéro du dernier enregistrement ou octet à verrouiller ou à déverrouiller.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les fonctions **Lock** et **Unlock** sont utilisées dans des environnements où plusieurs processus peuvent avoir besoin d'accéder

au même fichier.

Les fonctions **Lock** et **Unlock** sont toujours utilisées par paires. Les arguments des fonctions **Lock** et **Unlock** doivent être totalement identiques.

Si *Record*, *FromRecord* ou *ToRecord* ne sont pas indiqués, le verrouillage portera sur la totalité du fichier. Si *Record* seul est spécifié, seul l'enregistrement sera verrouillé/déverrouillé.

Si le fichier a été ouvert en mode d'entrée ou de sortie séquentielle, **Lock** et **Unlock** s'appliquent à l'ensemble du fichier, quelle que soit la plage spécifiée par *FromRecord* et *ToRecord*.

Exemple

Cet exemple illustre l'utilisation des fonctions **Lock** et **Unlock**. Nous supposons que `People.txt` est un fichier contenant les enregistrements de la structure `Person`.

VB

```
Structure Person
   Dim Name As String
   Dim ID As Integer
End Structure

Sub PutInLockedFile(ByVal index As Integer, ByVal onePerson As Person)
   Try
      FileOpen(1, "c:\people.txt", OpenMode.Binary)
      Lock(1)
      FilePut(index, onePerson)
      Unlock(1)
      FileClose(1)
   Catch
      ' Error recovery code here.
   End Try
End Sub
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileOpen, fonction

IOException Class

Autres ressources

Accès au fichier avec Visual Basic

LOF, fonction

Retourne une valeur de type **Long** représentant la taille, exprimée en octets, d'un fichier ouvert à l'aide de la fonction **FileOpen**.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **LOF**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Function LOF(ByVal FileNumber As Integer) As Long
```

Paramètres
FileNumber

Requis. Valeur de type **Integer** contenant un numéro de fichier valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Utilisez la fonction **FileLen** pour obtenir la longueur d'un fichier qui n'est pas ouvert.

Exemple

Cet exemple utilise la fonction **LOF** pour déterminer la taille d'un fichier ouvert. Nous supposons que `TestFile` est un fichier texte contenant des données exemple.

VB

```
Dim length As Long
FileOpen(1, "C:\TESTFILE.TXT", OpenMode.Input) ' Open file.
length = LOF(1)   ' Get length of file.
MsgBox(length)
FileClose(1)   ' Close file.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
EOF, fonction
FileLen, fonction
Loc, fonction
FileOpen, fonction
IOException Class
Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic

LSet, fonction

Retourne une chaîne alignée à gauche contenant la chaîne spécifiée ajustée à la longueur spécifiée.

```
Public Shared Function LSet( _
       ByVal Source As String, _
       ByVal Length As Integer _
) As String
```

Paramètres

Source

Requis. Expression **String**. Variable String à ajuster.

Length

Requis. Expression **Integer**. Longueur de chaîne retournée.

Notes

Si *Source* est plus long que *Length*, **LSet** retourne uniquement les caractères les plus à droite, jusqu'à atteindre la longueur de *Length*. Si *Source* est plus court que *Length*, **LSet** ajoute des espaces à l'extrémité droite de la chaîne retournée afin d'obtenir la longueur appropriée.

Exemple

Cet exemple illustre l'utilisation de la fonction **LSet**.

VB

```
Dim TestString As String = "Left"
Dim lString As String
' Returns "Left      "
lString = LSet(TestString, 10)
' Returns "Le"
lString = LSet(TestString, 2)
' Returns "Left"
lString = LSet(TestString, 4)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Liste des types de données (Visual Basic)
RSet, fonction

Concepts

Utilisation efficace des types de données

Fonctions M-R

Le tableau suivant contient une liste des fonctions membres runtime Visual Basic.

Mid	Minute	MIRR	MkDir
Month	MonthName	MsgBox	NPer
NPV	Oct	Partition	Pmt
PPmt	Print	PrintLine	PV
QBColor	Randomize	Rate	Rename
Replace	Reset	RGB	Right
RmDir	Rnd	RSet	RTrim

La fonction mathématique **Round**, qui était disponible en tant que fonction membre runtime dans les versions 6.0 et antérieures de Visual Basic, a été remplacée par une méthode (Round) de la classe Math de la bibliothèque de classes .NET Framework. Pour plus d'informations, consultez Fonctions mathématiques (Visual Basic).

Notes du développeur sur Smart Device

Les éléments suivants ne sont pas pris en charge dans les applications Smart Device : **MkDir**, **Print**, **PrintLine**, **Rename**, **Reset** et **RmDir**.

Voir aussi

Référence

Fonctions A-C
Fonctions D-G
Fonctions H-L
Fonctions S-Z

Autres ressources

Référence Visual Basic

Mid, fonction (Visual Basic)

Retourne une chaîne contenant un nombre spécifié de caractères d'une chaîne.

```
Public Shared Function Mid( _
   ByVal str As String, _
   ByVal Start As Integer, _
   Optional ByVal Length As Integer _
) As String
```

Paramètres

str

Requis. Expression **String** à partir de laquelle les caractères sont retournés.

Start

Requis. Expression **Integer**. Position de départ des caractères à retourner. Si l'argument *Start* est supérieur au nombre de caractères contenus dans l'argument *str*, la fonction **Mid** retourne une chaîne de longueur nulle (""). *Start* est de base un.

Length

Facultatif. Expression **Integer**. Nombre de caractères à retourner. Si cet argument est omis ou si le nombre de caractères dans le texte (y compris le caractère à la position *Start*) est inférieur à ceux qui sont compris dans l'argument *Length*, tous les caractères entre la position de début et la fin de la chaîne sont retournés.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Start* <= 0 ou *Length* < 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Pour déterminer le nombre de caractères contenus dans *str*, utilisez la fonction **Len**.

Visual Basic possède la fonction **Mid** et l'instruction **Mid**. Ces éléments fonctionnent à la fois sur un nombre spécifié de caractères dans une chaîne, mais la fonction **Mid** retourne les caractères pendant que l'instruction **Mid** remplace les caractères. Pour plus d'informations, consultez Mid, instruction.

Remarque

Dans les précédentes versions de Visual Basic, la fonction **MidB** retourne une chaîne en octets plutôt que des caractères. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Chara cter Set). Toutes les chaînes Visual Basic sont en Unicode, et la fonction **MidB** n'est plus prise en charge.

Exemple

L'exemple suivant utilise la fonction **Mid** pour retourner un nombre spécifié de caractères à partir d'une chaîne.

VB

```
' Creates text string.
Dim TestString As String = "Mid Function Demo"
' Returns "Mid".
Dim FirstWord As String = Mid(TestString, 1, 3)
' Returns "Demo".
Dim LastWord As String = Mid(TestString, 14, 4)
' Returns "Function Demo".
Dim MidWords As String = Mid(TestString, 5)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Left, fonction (Visual Basic)
Len, fonction (Visual Basic)
Fonctions Trim, LTrim et Rtrim
Mid, instruction
Right, fonction (Visual Basic)
ArgumentException

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

Autres ressources

Chaînes en Visual Basic
Introduction aux chaînes dans Visual Basic

Minute, fonction

Retourne une valeur **Integer** comprise entre 0 et 59 représentant la minute de l'heure.

```
Public Function Minute(ByVal TimeValue As DateTime) As Integer
```

Paramètres
TimeValue

Requis. Valeur **Date** de laquelle vous souhaitez extraire la minute.

Notes

Vous pouvez également obtenir la minute de l'heure en appelant **DatePart** et en spécifiant **DateInterval.Minute** pour l'argument *Interval*.

Exemple

Cet exemple utilise la fonction **Minute** pour obtenir la minute de l'heure à partir d'une heure spécifiée. Dans l'environnement de développement, le littéral d'heure est affiché sous le format d'heure abrégée en utilisant les paramètres régionaux de votre code.

VB

```
Dim thisTime As Date
Dim thisMinute As Integer
thisTime = #4:35:17 PM#
thisMinute = Minute(thisTime)
' thisMinute now contains 35.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Day, fonction (Visual Basic)
Hour, fonction (Visual Basic)
Now, propriété
Second, fonction (Visual Basic)
TimeOfDay, propriété
DatePart, fonction (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

MIRR, fonction

Retourne une donnée de type **Double** indiquant le taux de rendement interne modifié d'une série de liquidités périodiques (paiements et encaissements).

```
Function MIRR( _
   ByRef ValueArray() As Double, _
   ByVal FinanceRate As Double, _
   ByVal ReinvestRate As Double _
) As Double
```

Paramètres

ValueArray

Requis. Tableau de **Double** spécifiant les valeurs des liquidités. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement).

FinanceRate

Requis. **Double** spécifiant le taux d'intérêt payé pour couvrir le coût du financement.

ReinvestRate

Requis. **Double** spécifiant le taux d'intérêt perçu sur les gains tirés des sommes réinvesties.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Un rang de *ValueArray* n'est pas égal à 1, *FinanceRate* = -1 ou *ReinvestRate* = -1
DivideByZeroException	11	Une division par zéro s'est produite.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Le taux de rendement interne modifié est le taux de rendement interne lorsque les paiements et les encaissements sont soumis à des taux différents. La fonction **MIRR** prend en compte à la fois le coût de l'investissement (*FinanceRate*) et le taux d'intérêt perçu pour les sommes réinvesties (*ReinvestRate*).

Les arguments *FinanceRate* et *ReinvestRateargument*s sont des pourcentages exprimés sous forme de valeurs décimales. Par exemple, 12 pour cent est exprimé sous la forme 0,12.

La fonction **MIRR** utilise l'ordre des valeurs dans le tableau pour interpréter l'ordre des paiements et des encaissements. Veillez à entrer dans le bon ordre les valeurs représentant vos paiements et vos encaissements.

Exemple

Dans cet exemple, la fonction **MIRR** retourne le taux de rendement interne modifié d'une série de mouvements de trésorerie contenus dans le tableau `Values()`. `LoanAPR` représente l'intérêt de financement tandis que `InvAPR` représente le taux d'intérêt reçu sur le réinvestissement.

VB

```
' Define money format.
Dim MoneyFmt As String = "###,##0.00"
' Define percentage format.
Dim PercentFmt As String = "#0.00"

Dim values(4) As Double
' Business start-up costs.
values(0) = -70000
```

```
' Positive cash flows reflecting income for four successive years.
values(1) = 22000
values(2) = 25000
values(3) = 28000
values(4) = 31000

' Use the MIRR function to calculate the internal return rate.
' Set the loan rate.
Dim LoanAPR As Double = 0.1
' Set the reinvestment rate.
Dim InvAPR As Double = 0.12
' Calculate internal rate.
Dim RetRate As Double = MIRR(values, LoanAPR, InvAPR)
' Display internal return rate.
MsgBox("The modified internal rate of return for these cash flows is " & _
    Format(Math.Abs(RetRate) * 100, CStr(PercentFmt)) & "%.")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

IRR, fonction
NPV, fonction
Liste des mots clés financiers
ArgumentException
DivideByZeroException

MkDir, fonction

Crée un nouveau répertoire.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **MkDir**. Pour plus d'informations, consultez My.Computer.FileSystem.CreateDirectory, méthode.

```
Public Sub MkDir(ByVal Path As String)
```

Paramètres
Path

Requis. Expression **String** qui identifie le répertoire à créer. *Path* peut inclure le lecteur. Si aucun lecteur n'est spécifié, **MkDir** crée le nouveau répertoire du lecteur actif.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	52	*Path* n'est pas spécifié ou il est vide.
SecurityException	70	autorisation refusée ;
IOException	75	Le répertoire existe déjà.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Remarque de sécurité

Les autorisations d'E/S de lecture et d'écriture sur le fichier sont nécessaires pour créer un répertoire ou un dossier. Pour plus d'informations, consultez FileIOPermission et Autorisations d'accès du code.

Notes

Cette fonction crée tout nouveau répertoire spécifié dans *Path*, à moins qu'il n'existe déjà ou qu'une partie de *Path* ne soit pas valide. Le paramètre *Path* doit spécifier un chemin d'accès au répertoire, et non au fichier.

Exemple

Cet exemple utilise la fonction **MkDir** pour créer un répertoire. Si aucun lecteur n'est spécifié, le nouveau répertoire est créé sur le lecteur actif.

VB

```
' Make new directory.
MkDir("C:\TESTDIR")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Tâches

Comment : créer un répertoire dans Visual Basic

Référence

ChDir, fonction

CurDir, fonction

RmDir, fonction

ArgumentException Class

IOException Class

Month, fonction (Visual Basic)

Retourne une valeur **Integer** comprise entre 1 et 12 représentant le mois de l'année.

```
Public Function Month(ByVal DateValue As DateTime) As Integer
```

Paramètres
DateValue

Requis. Valeur **Date** de laquelle vous souhaitez extraire le mois.

Notes

Vous pouvez également obtenir le mois de l'année en appelant **DatePart** et en spécifiant **DateInterval.Month** pour l'argument *Interval*.

Exemple

Cet exemple utilise la fonction **Month** pour obtenir le mois à partir d'une date spécifiée. Dans l'environnement de développement, le littéral de date est affiché sous le format de date courte en utilisant les paramètres régionaux de votre code.

VB

```
Dim thisDate As Date
Dim thisMonth As Integer
thisDate = #2/12/1969#
thisMonth = Month(thisDate)
' thisMonth now contains 2.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Day, fonction (Visual Basic)
Now, propriété
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

MonthName, fonction (Visual Basic)

Retourne une valeur **String** contenant le nom du mois spécifié.

```
Public Function MonthName( _
   ByVal Month As Integer, _
   Optional ByVal Abbreviate As Boolean = False _
) As String
```

Paramètres

Month

Requis. **Integer**. Désignation numérique du mois, entre 1 et 13 ; 1 représente janvier et 12, décembre. Vous pouvez utiliser la valeur 13 avec un calendrier de 13 mois. Si votre système utilise un calendrier de 12 mois et que *Month* a la valeur 13, **MonthName** retourne une chaîne vide.

Abbreviate

Facultatif. Valeur **Boolean** qui indique si le nom du mois doit être abrégé. Si cette valeur est omise, la valeur par défaut est **False**, ce qui signifie que le nom du mois n'est pas abrégé.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Month* est inférieur à 1 ou supérieur à 13.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La chaîne retournée par **MonthName** dépend non seulement des arguments d'entrée, mais également des paramètres **Options régionales** spécifiés dans le **Panneau de configuration** Windows.

Remarque

MonthName utilise le paramètre de calendrier actuel de la propriété CurrentCulture de la classe CultureInfo dans l'espace de noms System.Globalization. Les valeurs par défaut de **CurrentCulture** dépendent des paramètres du **Panneau de configuration**.

Exemple

Cet exemple utilise la fonction **MonthName** pour déterminer le nom du mois, en fonction de l'entier spécifié. La valeur Boolean détermine si le nom complet (**False**) ou abrégé (**True**) est affiché.

VB

```
Dim thisMonth As Integer
Dim name As String
thisMonth = 4
' Set Abbreviate to True to return an abbreviated name.
name = MonthName(thisMonth, True)
' name now contains "Apr".
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

WeekdayName, fonction (Visual Basic)

System

System.Globalization

DateTime

ArgumentException

CultureInfo

MsgBox, fonction (Visual Basic)

Affiche un message dans une boîte de dialogue, attend que l'utilisateur clique sur un bouton, puis retourne un nombre entier qui indique le bouton choisi par l'utilisateur.

```
Public Function MsgBox( _
   ByVal Prompt As Object, _
   Optional ByVal Buttons As MsgBoxStyle = MsgBoxStyle.OKOnly, _
   Optional ByVal Title As Object = Nothing _
) As MsgBoxResult
```

Paramètres

Prompt

Requis. Expression **String** affichée comme message de la boîte de dialogue. La longueur maximale de l'argument *Prompt* est d'environ 1 024 caractères selon la largeur des caractères utilisés. Si *Prompt* se compose de plusieurs lignes, vous pouvez séparer les lignes à l'aide d'un retour chariot (13 **Chr( )**), d'un saut de ligne (10 **Chr( )**) ou d'une combinaison des deux caractères (13 **Chr( )**& 10 **Chr( )**).

Buttons

Facultatif. Expression numérique qui représente la somme des valeurs spécifiant le nombre et le type de boutons à afficher, le style d'icône à utiliser, l'identité du bouton par défaut, ainsi que la modalité du message. Si l'argument *Buttons* est omis, la valeur par défaut est zéro.

Title

Facultatif. Expression **String** affichée dans la barre de titre de la boîte de dialogue. Si l'argument *Title* est omis, le nom de l'application est placé dans la barre de titre.

Paramètres

Les valeurs d'énumération **MsgBoxStyle** sont répertoriées dans le tableau suivant.

Membre	Valeur	Description
OKOnly	0	Affiche le bouton OK uniquement.
OKCancel	1	Affiche les boutons OK et Annuler.
AbortRetryIgnore	2	Affiche les boutons Abandonner, Réessayer et Ignorer.
YesNoCancel	3	Affiche les boutons Oui, Non et Annuler.
YesNo	4	Affiche les boutons Oui et Non.
RetryCancel	5	Affiche les boutons Réessayer et Annuler.
Critical	16	Affiche l'icône Message critique.
Question	32	Affiche l'icône Requête d'avertissement.
Exclamation	48	Affiche l'icône Message d'avertissement.
Information	64	Affiche l'icône Message d'information.
DefaultButton1	0	Le premier bouton est le bouton par défaut.
DefaultButton2	256	Le deuxième bouton est le bouton par défaut.

DefaultButton3	512	Le troisième bouton est le bouton par défaut.
ApplicationModal	0	L'application est modale. L'utilisateur doit répondre au message avant de poursuivre le travail dans l'application en cours.
SystemModal	4096	Le système est modal. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message.
MsgBoxSetForeground	65536	Spécifie la fenêtre de message comme fenêtre de premier plan.
MsgBoxRight	524288	Le texte est aligné à droite.
MsgBoxRtlReading	1048576	Spécifie que le texte doit apparaître de droite à gauche sur les systèmes hébraïques et arabes.

Le premier groupe de valeurs (0-5) décrit le nombre et le type des boutons affichés dans la boîte de dialogue. Le second groupe (16, 32, 48, 64) décrit le style de l'icône. Le troisième groupe (0, 256, 512) détermine quel bouton est la valeur par défaut. Le quatrième groupe (0, 4096) détermine la modalité du message, et le cinquième groupe indique si la fenêtre de message est la fenêtre de premier plan, avec l'alignement et l'orientation du texte. Au moment d'additionner ces nombres pour obtenir la valeur finale de l'argument *Buttons*, ne sélectionnez qu'un seul nombre dans chaque groupe.

Valeur de retour

Constante	Valeur
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Prompt* n'est pas une expression **String** ou *Title* n'est pas valide.
InvalidOperationException	5	Le processus ne s'exécute pas en mode utilisateur interactif.
InvalidEnumArgumentException	5	Au moins un paramètre n'est pas un membre de l'énumération **MsgBoxResult** ou **MsgBoxStyle**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si la boîte de dialogue est dotée d'un bouton **Annuler**, appuyer sur Échap équivaut à cliquer sur **Annuler**. Si la boîte de

dialogue contient un bouton **Aide**, cela signifie qu'une aide contextuelle relative à la boîte de dialogue est disponible. Toutefois, aucune valeur n'est retournée tant que l'utilisateur n'a pas cliqué sur l'un des autres boutons.

Remarque

Si vous souhaitez préciser plus d'informations que celles fournies par le premier argument, vous devez utiliser la fonction **MsgBox** dans une expression. Si vous omettez certains arguments de position, vous devez quand même conserver la virgule de séparation correspondante.

Remarque

La fonction **MsgBox** nécessite **UIPermission** au niveau SafeTopLevelWindows, ce qui peut affecter son exécution dans les situations où le niveau de confiance n'est pas total. Pour plus d'informations, consultez Demande d'autorisations et UIPermission.

Exemple

Cet exemple utilise la fonction **MsgBox** pour afficher un message d'erreur grave dans une boîte de dialogue pourvue des boutons Yes et No. Le bouton No est spécifié comme réponse par défaut. Ce résultat est obtenu en combinant les valeurs des constantes **MsgBox** dans une expression numérique. Dans ce cas, l'addition de 4 (la combinaison des boutons Oui/Non) avec 16 (la fenêtre **Message critique**) et 256 (le deuxième bouton comme bouton par défaut) donne un total de 276. La valeur retournée par la fonction **MsgBox** dépend du bouton choisi par l'utilisateur : Oui retourne une valeur égale à 6 ; Non retourne une valeur égale à 7.

VB

```
Dim msg As String
Dim title As String
Dim style As MsgBoxStyle
Dim response As MsgBoxResult
msg = "Do you want to continue?"    ' Define message.
style = MsgBoxStyle.DefaultButton2 Or _
   MsgBoxStyle.Critical Or MsgBoxStyle.YesNo
title = "MsgBox Demonstration"    ' Define title.
' Display message.
response = MsgBox(msg, style, title)
If response = MsgBoxResult.Yes Then    ' User chose Yes.
   ' Perform some action.
Else
   ' Perform some other action.
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

InputBox, fonction (Visual Basic)

NPer, fonction

Retourne un **Double** spécifiant le nombre d'échéances d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.

```
Function NPer( _
   ByVal Rate As Double, _
   ByVal Pmt As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

Paramètres

Rate

Requis. **Double** spécifiant le taux d'intérêt par période. Par exemple, si vous obtenez un prêt pour l'achat d'une voiture à un taux d'intérêt annuel (TIA) de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

Pmt

Requis. **Double** spécifiant le versement à effectuer à chaque échéance. Les paiements comprennent généralement le capital et les intérêts qui ne sont pas modifiés au cours de l'annuité.

PV

Requis. **Double** spécifiant la valeur actuelle, ou valeur au jour d'aujourd'hui, d'une série de paiements ou d'encaissements futurs. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond pour le prêteur à la valeur actuelle des remboursements mensuels que vous allez effectuer pour la voiture.

FV

Facultatif. **Double** spécifiant la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 €, car il s'agit de sa valeur après le dernier remboursement. Toutefois, si vous souhaitez économiser 50 000 F sur 18 ans pour les études de votre enfant, ce montant constitue la valeur future. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être `DueDate.EndOfPeriod` si les paiements sont dus à terme échu ou `DueDate.BegOfPeriod` si les paiements sont dus à terme à échoir. Si cet argument est omis, `DueDate.EndOfPeriod` est utilisé par défaut.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Rate* <= -1.
ArgumentException	5	*Rate* = 0 et *Pmt* = 0

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Exemple

Cet exemple utilise la fonction **NPer** pour retourner le nombre de périodes nécessaires pour rembourser un emprunt dont le montant est représenté par `PVal`. Sont également spécifiés le taux d'intérêt par période (`APR / 12`), le montant des paiements (`Payment`), la valeur future de l'emprunt (`FVal`), ainsi qu'un nombre indiquant si les remboursements s'effectuent au début ou à la fin des échéances (`PayType`).

VB

```
Sub TestNPer()
    Dim TotPmts As Double
    Dim PVal, APR, Payment As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Usually 0 for a loan.
    Dim Fval As Double = 0
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    ' Usually 0 for a loan.
    If APR > 1 Then APR = APR / 100
    Payment = CDbl(InputBox("How much do you want to pay each month?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    TotPmts = NPer(APR / 12, -Payment, PVal, FVal, PayType)
    If Int(TotPmts) <> TotPmts Then TotPmts = Int(TotPmts) + 1

    MsgBox("It will take you " & TotPmts & " months to pay off your loan.")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

ArgumentException

NPV, fonction

Retourne une valeur de type **Double** indiquant la valeur nette actuelle d'un investissement, calculée en fonction d'une série de liquidités périodiques (paiements et encaissements) et d'un taux d'escompte.

```
Function NPV( _
   ByVal Rate As Double, _
   ByRef ValueArray() As Double _
) As Double
```

Paramètres

Rate

Requis. Donnée de type **Double** indiquant le taux d'escompte sur la période, exprimé sous la forme d'un nombre décimal.

ValueArray

Requis. Tableau de valeurs **Double** représentant les valeurs des liquidités. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement).

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*ValueArray* a la valeur **Nothing**, le rang de *ValueArray* <> 1 ou *Rate* = -1

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La valeur nette actuelle d'un investissement est la valeur actuelle d'une série de paiements et d'encaissements futurs.

La fonction **NPV** utilise l'ordre des valeurs dans le tableau pour interpréter l'ordre des paiements et des encaissements. Veillez à entrer dans le bon ordre les valeurs représentant vos paiements et vos encaissements.

La fonction **NPV** fait débuter l'investissement une échéance avant la date de valeur du premier mouvement de trésorerie et le fait se terminer avec le dernier mouvement de trésorerie du tableau.

Le calcul de la valeur nette actuelle est basé sur les liquidités futures. Si votre premier mouvement de trésorerie se produit au début de la première échéance, la première valeur doit alors être ajoutée à la valeur retournée par la fonction **NPV** et ne doit pas être incluse dans les valeurs de mouvements de trésorerie de *ValueArray*.

La fonction **NPV** est similaire à la fonction **PV** (valeur actuelle) à la différence que la fonction **PV** autorise le début des mouvements de trésorerie soit à la fin, soit au début d'une échéance. Contrairement aux valeurs de mouvements de trésorerie variables de la fonction **NPV**, les mouvements de trésorerie de la fonction **PV** doivent être constants tout au long de la période d'investissement.

Exemple

Cet exemple utilise la fonction **NPV** pour retourner la valeur nette actuelle pour des liquidités listées dans le tableau `values()`. La valeur de retour, stockée dans `FixedRetRate`, représente le taux de rendement interne fixe.

VB

```
' Define money format.
Dim MoneyFmt As String = "###,##0.00"
' Define percentage format.
Dim PercentFmt As String = "#0.00"

Dim values(4) As Double
' Business start-up costs.
values(0) = -70000
' Positive cash flows reflecting income for four successive years.
```

```
values(1) = 22000
values(2) = 25000
values(3) = 28000
values(4) = 31000

' Use the NPV function to calculate the net present value.
' Set fixed internal rate.
Dim FixedRetRate As Double = 0.0625
' Calculate net present value.
Dim NetPVal As Double = NPV(FixedRetRate, values)
' Display net present value.
MsgBox("The net present value of these cash flows is " & _
    Format(NetPVal, MoneyFmt) & ".")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

IRR, fonction

MIRR, fonction

Liste des mots clés financiers

ArgumentException

Oct, fonction

Retourne une chaîne représentant la valeur octale d'un nombre.

```
Public Shared Function Oct( _
   ByVal Number As { Byte | SByte | Short | UShort | _
   Integer | UInteger | Long | ULong | Object } _
) As String
```

Paramètres

Number

Requis. Toute expression numérique ou expression **String** valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	5	*Number* n'est pas spécifié.
ArgumentException	5	*Number* n'est pas un type numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si *Number* n'est pas déjà un nombre entier, il est arrondi au nombre entier le plus proche avant d'être évalué.

Si Number est	Oct retourne
Empty	Zéro (0)
Tout autre nombre	22 caractères octaux maximum

Vous pouvez représenter des nombres octaux directement en faisant précéder les nombres de la plage appropriée de **&O**. Par exemple, `&O10` est la notation octale du 8 décimal.

Exemple

L'exemple suivant utilise la fonction **Oct** pour retourner la valeur octale d'un nombre.

VB

```
Dim TestOct As String
' Returns "4".
TestOct = Oct(4)
' Returns "10".
TestOct = Oct(8)
' Returns "713".
TestOct = Oct(459)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des conversions
Hex, fonction (Visual Basic)
Fonctions de conversion de types de données
ArgumentNullException

Partition, fonction

Retourne une chaîne représentant la plage calculée qui contient un nombre.

```
Public Function Partition( _
   ByVal Number As Long, _
   ByVal Start As Long, _
   ByVal Stop As Long, _
   ByVal Interval As Long _
) As String
```

Paramètres

Number

Requis. **Long**. Nombre entier que vous souhaitez localiser dans l'une des plages calculées.

Start

Requis. **Long**. Nombre entier qui indique le démarrage du jeu de plages calculées. *Start* ne peut pas être inférieur à 0.

Stop

Requis. **Long**. Nombre entier qui indique la fin du jeu de plages calculées. *Stop* ne peut pas être inférieur ou égal à *Start*.

Interval

Requis. **Long**. Nombre entier qui indique la taille de chaque plage calculée entre *Start* et *Stop*. *Interval* ne peut pas être inférieur à 1.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Start* < 0, *Stop* <= *Start*, ou *Interval* < 1.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Partition** calcule un ensemble de plages numériques, chacune contenant le nombre de valeurs spécifiées par *Interval*. La première plage commence à *Start* et la dernière se termine à *Stop*. La fonction **Partition** identifie ensuite la plage qui contient *Number* et retourne une chaîne décrivant cette plage. La plage est représentée dans la chaîne par "*lowervalue*:*uppervalue*", où la valeur la plus basse (*lowervalue*) de la plage est séparée de la valeur la plus élevée (*uppervalue*) par un signe deux-points (:).

Au besoin, la fonction **Partition** insère des espaces à gauche avant *lowervalue* et *uppervalue* de telle sorte que les deux arguments possèdent le même nombre de caractères que la représentation de chaîne de la valeur (*Stop* + 1). Ainsi, si vous utilisez la sortie de la fonction **Partition** avec plusieurs valeurs de *Number*, le texte obtenu pourra être correctement géré au cours d'un tri ultérieur éventuel.

Le tableau ci-dessous illustre certains exemples de chaînes pour des plages calculées à partir de trois jeux d'arguments *Start*, *Stop*, et *Interval*. Les colonnes "Première plage" et "Dernière plage" illustrent les plus petites et les plus grandes plages possibles attribuées aux valeurs de *Start* et *Stop*. Les colonnes "Avant première plage" et "Après dernière plage" indiquent les chaînes retournées pour les valeurs de *Number* inférieures à *Start* et supérieures à *Stop*, respectivement.

Start	*Stop*	*Interval*	Avant première plage	Première plage	Dernière plage	Après dernière plage
0	99	5	" : -1"	" 0: 4"	" 95: 99"	"100: "
20	199	10	" : 19"	" 20: 29"	"190:199"	"200: "

100	1010	20	" : 99"	" 100: 119"	"1000:1010"	"1011: "

Dans ce tableau, la troisième ligne illustre le résultat obtenu lorsque *Start* et *Stop* définissent un ensemble de nombres qui ne peut être divisé de façon égale par *Interval*. La dernière plage se termine à *Stop*, contient seulement 11 nombres, même si *Interval* est égal à 20.

Si *Interval* est égal à 1, la plage est "*Number*:*Number*", indépendamment des arguments *Start* et *Stop*. Par exemple, si *Number* est égal à 267, *Stop* à 1 000 et *Interval* à 1, **Partition** retourne " 267: 267".

Partition peut être utile lors de la création de requêtes de base de données. Vous pouvez créer une requête SELECT qui indique le nombre de commandes appartenant à diverses plages (par exemple, les commandes dont la valeur de facturation est comprise entre 1 et 1 000, entre 1 001 et 2 000, etc.).

Exemple

L'exemple suivant configure une série de plages pour les décennies de 1950 à 2049. Il localise la valeur de `year` dans la plage appropriée et retourne une valeur **String** qui affiche la plage. Si `year` a une valeur de 1984, par exemple, **Partition** retourne à "1980:1989".

VB

```
Dim year As Long = 1984
' Assume the value of year is provided by data or by user input.
Dim decade As String
decade = Partition(year, 1950, 2049, 10)
MsgBox("Year " & CStr(year) & " is in decade " & decade & ".")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Autres ressources

Comment faire (SQL Server)

Pmt, fonction

Retourne une valeur de type **Double** indiquant le montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.

```
Function Pmt( _
   ByVal Rate As Double, _
   ByVal NPer As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

Paramètres

Rate

Requis. **Double** spécifie le taux d'intérêt par période. Par exemple, si vous obtenez un prêt pour l'achat d'une voiture à un taux d'intérêt annuel (TIA) de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

NPer

Requis. **Double** spécifie le nombre total d'échéances de l'annuité. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt d'une durée de quatre ans pour l'achat d'une voiture, votre emprunt comporte un total de 4 x 12 (soit 48) échéances.

PV

Requis. **Double** spécifie la valeur actuelle (ou montant forfaitaire) d'une série de paiements devant être effectués dans le futur. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond pour le prêteur à la valeur actuelle des remboursements mensuels que vous allez effectuer pour la voiture.

FV

Facultatif. **Double** indique la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 €, car il s'agit de sa valeur après le dernier remboursement. Toutefois, si vous souhaitez économiser 50 000 € sur 18 ans pour les études de votre enfant, ce montant constitue la valeur future. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être `DueDate.EndOfPeriod` si les paiements sont dus à terme échu ou `DueDate.BegOfPeriod` si les paiements sont dus à terme à échoir. Si cet argument est omis, `DueDate.EndOfPeriod` est utilisé par défaut.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*NPer* = 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs (vous pouvez comparer le numéro d'erreur et la propriété Number (objet Err)). Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Les arguments *Rate* et *NPer* doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités. Par exemple, si *Rate* est exprimé en mois, il doit en être de même pour *NPer*.

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Exemple

Cet exemple utilise la fonction **Pmt** pour retourner le remboursement mensuel pour un emprunt sur une période fixe. Sont spécifiés le taux d'intérêt par période (`APR / 12`), le nombre total de versements (`TotPmts`), la valeur actuelle ou le capital de l'emprunt (`PVal`), la valeur future de l'emprunt (`FVal`) ainsi qu'un nombre indiquant si les remboursements s'effectuent au début ou à la fin des échéances `(PayType)`.

VB

```
Sub TestPMT()
    Dim PVal, APR, Payment, TotPmts As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    ' Usually 0 for a loan.
    Dim FVal As Double = 0
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    If APR > 1 Then APR = APR / 100 ' Ensure proper form.
    TotPmts = CDbl(InputBox("How many monthly payments will you make?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    Payment = Pmt(APR / 12, TotPmts, -PVal, FVal, PayType)

    MsgBox("Your payment will be " & Format(Payment, Fmt) & " per month.")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

ArgumentException

PPmt, fonction

Retourne une valeur de type **Double** indiquant le remboursement du capital, pour une période donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.

```
Function PPmt( _
   ByVal Rate As Double, _
   ByVal Per As Double, _
   ByVal NPer As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

Paramètres

Rate

Requis. **Double** spécifie le taux d'intérêt par période. Par exemple, si vous obtenez un prêt pour l'achat d'une voiture à un taux d'intérêt annuel (TIA) de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

Per

Requis. **Double** indique la périodicité des paiements, comprise entre 1 et *NPer*.

NPer

Requis. **Double** spécifie le nombre total d'échéances de l'annuité. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt d'une durée de quatre ans pour l'achat d'une voiture, votre emprunt comporte un total de 4 x 12 (soit 48) échéances.

PV

Requis. **Double** spécifie la valeur actuelle d'une série de paiements ou recettes futurs. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond pour le prêteur à la valeur actuelle des remboursements mensuels que vous allez effectuer pour la voiture.

FV

Facultatif. **Double** indique la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 €, car il s'agit de sa valeur après le dernier remboursement. Toutefois, si vous souhaitez économiser 50 000 F sur 18 ans pour les études de votre enfant, ce montant constitue la valeur future. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être `DueDate.EndOfPeriod` si les paiements sont dus à terme échu ou `DueDate.BegOfPeriod` si les paiements sont dus à terme à échoir. Si cet argument est omis, `DueDate.EndOfPeriod` est utilisé par défaut.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Per* <=0 ou *Per* > *NPer*.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Les arguments *Rate* et *NPer* doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités. Par exemple, si *Rate* est exprimé en mois, il doit en être de même pour *NPer*.

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Exemple

Cet exemple utilise la fonction **PPmt** pour déterminer la proportion d'un versement pour une période donnée correspondant au remboursement du capital, sachant que le montant des remboursements reste constant. Sont spécifiés le taux d'intérêt par période (`APR / 12`), la période de remboursement pour laquelle on souhaite connaître la proportion du capital (`Period`), le nombre total des remboursements (`TotPmts`), la valeur actuelle ou le capital de l'emprunt (`PVal`), la valeur future de l'emprunt (`FVal`) et un nombre indiquant si les remboursements s'effectuent au début ou à la fin des échéances (`PayType`).

VB

```
Sub TestPPMT()
    Dim PVal, APR, TotPmts, Payment, Period, P, I As Double
    Dim PayType As DueDate
    Dim Msg As String
    Dim Response As MsgBoxResult

    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    ' Usually 0 for a loan.
    Dim Fval As Double = 0
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    ' Ensure proper form.
    If APR > 1 Then APR = APR / 100
    TotPmts = CDbl(InputBox("How many monthly payments do you have to make?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    Payment = Math.Abs(-Pmt(APR / 12, TotPmts, PVal, FVal, PayType))
    Msg = "Your monthly payment is " & Format(Payment, Fmt) & ". "
    Msg = Msg & "Would you like a breakdown of your principal and "
    Msg = Msg & "interest per period?"
    ' See if chart is desired.
    Response = MsgBox(Msg, MsgBoxStyle.YesNo)
    If Response <> MsgBoxResult.No Then
        If TotPmts > 12 Then MsgBox("Only first year will be shown.")
        Msg = "Month  Payment  Principal  Interest" & vbNewLine
        For Period = 1 To TotPmts
            ' Show only first 12.
            If Period > 12 Then Exit For
            P = PPmt(APR / 12, Period, TotPmts, -PVal, FVal, PayType)
            ' Round principal.
            P = (Int((P + 0.005) * 100) / 100)
            I = Payment - P
            ' Round interest.
            I = (Int((I + 0.005) * 100) / 100)
            Msg = Msg & Period & vbTab & Format(Payment, Fmt)
            Msg = Msg & vbTab & Format(P, Fmt) & vbTab & Format(I, Fmt) & vbNewLine
        Next Period
        ' Display amortization table.
        MsgBox(Msg)
    End If
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

ArgumentException

Fonctions Print, PrintLine

Écrit des données mises en forme pour l'affichage dans un fichier séquentiel.

```
Public Sub Print( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output() As Object _
)
' -or-
Public Sub PrintLine( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output() As Object _
)
```

Paramètres

FileNumber

Requis. Tout numéro de fichier valide.

Output

Facultatif. Zéro ou plusieurs expressions séparées par des virgules à écrire dans un fichier.

L'argument *Output* peut prendre les valeurs suivantes :

Paramètre	**Description**
SPC(*n* **)**	Utilisé pour insérer des espaces dans la sortie, *n* indiquant le nombre d'espaces à insérer.
TAB(*n* **)**	Utilisé pour placer le point d'insertion sur un numéro de colonne absolu, *n* indiquant le numéro de colonne. Utilisez **TAB** sans argument pour positionner le point d'insertion au début de la zone d'impression suivante.
expression	Expressions numériques ou expressions de chaîne à imprimer.

Exceptions

Type d'exception	**Numéro de l'erreur**	**Condition**
IOException	54	Le mode de fichier est non valide.
IOException	52	*FileNumber* n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les fonctions **Print** et **PrintLine** sont fournies pour une compatibilité descendante et peut avoir un impact sur les performances. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Print n'inclut pas de saut de ligne à la fin d'une ligne ; cependant **PrintLine** inclut un saut de ligne.

Les données écrites à l'aide de **Print** sont généralement lues à partir d'un fichier avec **LineInput** ou **Input**.

Si l'argument *Output* est omis pour **PrintLine**, une ligne vide est imprimée dans le fichier ; pour **Print**, aucune sortie n'est effectuée. Plusieurs expressions séparées par une virgule seront alignées sur les limites de tabulation, mais la combinaison de virgules et de **TAB** peut engendrer des résultats incohérents.

Pour les données de type **Boolean**, les valeurs `True` ou `False` sont imprimées. Les mots clés **True** et **False** ne sont jamais traduits, quels que soient les paramètres régionaux.

Les données relatives aux dates sont écrites dans le fichier selon le format de date courte standard reconnu par votre système. Si l'un des composants date ou heure est omis ou égal à zéro, seul le composant fourni est écrit dans le fichier.

Si les données de *Output* sont vides, rien n'est écrit dans le fichier. Toutefois si les données de la liste *Output* sont de type **DBNull**, la valeur `Null` est écrite dans le fichier.

Pour les données de type **Error**, le résultat affiché est `Error errorcode`. Le mot clé **Error** n'est jamais traduit, quels que soient les paramètres régionaux.

Toutes les données écrites dans le fichier à l'aide de la fonction **Print** respectent les conventions internationales ; autrement dit, les données sont mises en forme à l'aide du séparateur décimal approprié. Si l'utilisateur souhaite extriaire des données en vue d'une utilisation par plusieurs paramètres régionaux, il convient d'utiliser la fonction **Write**.

L'écriture sur fichier à l'aide de la fonction **Print** ou **PrintLine** requiert l'accès **Write** à partir de l'énumération **FileIOPermissionAccess**. Pour plus d'informations, consultez FileIOPermissionAccess, énumération.

Exemple

Cet exemple utilise les fonctions **Print** et **PrintLine** pour écrire des données dans un fichier.

VB

```
FileOpen(1, "c:\trash.txt", OpenMode.Output)   ' Open file for output.
Print(1, "This is a test.")   ' Print text to file.
PrintLine(1)   ' Print blank line to file.
PrintLine(1, "Zone 1", TAB(), "Zone 2")   ' Print in two print zones.
PrintLine(1, "Hello", "World")   ' Separate strings with a tab.
PrintLine(1, SPC(5), "5 leading spaces ")   ' Print five leading spaces.
PrintLine(1, TAB(10), "Hello")   ' Print word at column 10.

' Assign Boolean, Date, and Error values.
Dim aBool As Boolean
Dim aDate As DateTime
aBool = False
aDate = DateTime.Parse("February 12, 1969")

' Dates and booleans are translated using locale settings of your system.
PrintLine(1, aBool, " is a Boolean value")
PrintLine(1, aDate, " is a date")
FileClose(1)   ' Close file.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : insérer du texte dans des fichiers dans Visual Basic

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Référence

FileOpen, fonction

SPC, fonction

TAB, fonction

Fonctions Write, WriteLine

Autres ressources

Accès au fichier avec Visual Basic

PV, fonction

Retourne une valeur de type **Double** indiquant le montant actuel d'une annuité basée sur des échéances futures constantes et périodiques et sur un taux d'intérêt fixe.

```
Function PV( _
   ByVal Rate As Double, _
   ByVal NPer As Double, _
   ByVal Pmt As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

Paramètres

Rate

Requis. **Double** spécifie le taux d'intérêt par période. Par exemple, si vous obtenez un prêt pour l'achat d'une voiture à un taux d'intérêt annuel (TIA) de 10 pour cent et si vous effectuez des remboursements mensuels, le taux par échéance est de 0,1/12, soit 0,0083.

NPer

Requis. **Double** spécifie le nombre total d'échéances de l'annuité. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt d'une durée de quatre ans pour l'achat d'une voiture, votre emprunt comporte 4 x 12 (soit 48) échéances.

Pmt

Requis. **Double** indique le versement à effectuer à chaque échéance. Les paiements comprennent généralement le capital et les intérêts qui ne sont pas modifiés au cours de l'annuité.

FV

Facultatif. **Double** indique la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 €, car il s'agit de sa valeur après le dernier remboursement. Toutefois, si vous souhaitez économiser 50 000 F sur 18 ans pour les études de votre enfant, ce montant constitue la valeur future. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être `DueDate.EndOfPeriod` si les paiements sont dus à terme échu ou `DueDate.BegOfPeriod` si les paiements sont dus à terme à échoir. Si cet argument est omis, `DueDate.EndOfPeriod` est utilisé par défaut.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Les arguments *Rate* et *NPer* doivent être calculés à l'aide d'échéances exprimées dans les mêmes unités. Par exemple, si *Rate* est exprimé en mois, il doit en être de même pour *NPer*.

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Exemple

Dans cet exemple, la fonction **PV** retourne la valeur actuelle d'une rente d'un million de francs qui produira un revenu de 50 000 F par an pendant 20 ans. Sont spécifiés le taux de rendement annuel escompté (`APR`), le nombre total des versements (`TotPmts`), le montant de chaque versement (`YrIncome`), la valeur future totale de l'investissement (`FVal`), ainsi qu'un nombre indiquant si les versements s'effectuent au début ou à la fin des échéances (`PayType`). Notez que, dans la mesure où `YrIncome` correspond à un prélèvement effectué chaque année sur le capital, il s'agit d'une valeur négative.

VB

```
Sub TestPV()
```

```
    ' Define money format.
    Dim Fmt As String = "###,##0.00"
    ' Annual percentage rate.
    Dim APR As Double = 0.0825
    ' Total number of payments.
    Dim TotPmts As Double = 20
    ' Yearly income.
    Dim YrIncome As Double = 50000
    ' Future value.
    Dim FVal As Double = 1000000
    ' Payment at beginning of month.
    Dim PayType As DueDate = DueDate.BegOfPeriod
    Dim PVal As Double = PV(APR, TotPmts, -YrIncome, FVal, PayType)
    MsgBox("The present value is " & Format(PVal, Fmt) & ".")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

QBColor, fonction

Retourne une valeur **Integer** représentant le code couleur RVB correspondant au numéro de couleurs spécifié.

```
QBColor(Color)
```

Paramètres
Color

Requis. Nombre entier compris entre 0 et 15.

Paramètres

L'argument *Color* possède les paramètres suivants :

Numéro	Couleur	Numéro	Couleur
0	Noir	8	Gris
1	Bleu	9	Bleu clair
2	Vert	10	Vert clair
3	Cyan	11	Cyan clair
4	Rouge	12	Rouge clair
5	Magenta	13	Magenta clair
6	Jaune	14	Jaune clair
7	Blanc	15	Blanc brillant

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Color* est en dehors de la plage comprise entre 0 et 15 inclus.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Commençant par l'octet le moins significatif, la valeur retournée spécifie les valeurs rouge, vert et bleu utilisées pour définir la couleur appropriée dans le système RVB utilisé par le langage Visual Basic.

Exemple

Cet exemple utilise la fonction **QBColor** pour utiliser la couleur indiquée par `colorInteger`. **QBColor** accepte des valeurs entières entre 0 et 15.

VB

```
Dim colorInteger As Integer
' Use 4 for red.
colorInteger = QBColor(4)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
RGB, fonction (Visual Basic)
ArgumentException Class

Randomize, fonction (Visual Basic)

Initialise le générateur de nombres aléatoires.

```
Public Shared Sub Randomize ([ Number ])
```

Paramètres
Number

Facultatif. **Object** ou expression numérique valide.

Notes

Randomize utilise l'élément *Number* pour initialiser le générateur de nombres aléatoires de la fonction **Rnd** en lui donnant une nouvelle valeur initiale. Si l'élément *Number* est omis, la valeur retournée par l'horloge système sert de nouvelle valeur de départ.

En l'absence de **Randomize**, la fonction **Rnd** (sans argument) adopte le même nombre comme valeur initiale la première fois qu'elle est appelée. Lors des appels ultérieurs, elle réutilise le dernier nombre généré comme valeur initiale.

Remarque

Pour répéter des séquences de nombres aléatoires, appelez **Rnd** avec un argument négatif juste avant d'utiliser **Randomize** avec un argument numérique. Si vous utilisez l'instruction **Randomize** en attribuant la même valeur à *Number*, la séquence précédente n'est pas répétée.

Remarque de sécurité

Étant donné que l'instruction **Random** et que la fonction **Rnd** commencent par une valeur initiale et génèrent des nombres c ompris dans une plage finie, les résultats peuvent être prévus par toute personne connaissant l'algorithme utilisé pour les cr éer. L'instruction **Random** et la fonction **Rnd** ne doivent donc pas être utilisées pour créer des nombres aléatoires qui seront utilisés en chiffrement. Pour plus d'informations, consultez RandomNumberGenerator.

Exemple

Cet exemple utilise l'instruction **Randomize** pour initialiser le générateur de nombres aléatoires. Compte tenu que l'argument number a été omis, **Randomize** utilise la valeur de retour à partir de la fonction **Timer** comme nouvelle valeur de départ.

VB

```
' Initialize the random-number generator.
Randomize()
' Generate random value between 1 and 6.
Dim value As Integer = CInt(Int((6 * Rnd()) + 1))
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : VBMath

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Liste des mots clés mathématiques
Rnd, fonction (Visual Basic)

Rate, fonction

Retourne une valeur de type **Double** indiquant le taux d'intérêt par échéance pour une annuité.

```
Function Rate( _
   ByVal NPer As Double, _
   ByVal Pmt As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod, _
   Optional ByVal Guess As Double = 0.1 _
) As Double
```

Paramètres

NPer

Requis. **Double** spécifie le nombre total d'échéances de l'annuité. Par exemple, si vous effectuez des remboursements mensuels dans le cadre d'un emprunt d'une durée de quatre ans pour l'achat d'une voiture, votre emprunt comporte un total de 4 * 12 (soit 48) échéances.

Pmt

Requis. **Double** indique le versement à effectuer à chaque échéance. Les paiements comprennent généralement le capital et les intérêts qui ne sont pas modifiés au cours de l'annuité.

PV

Requis. **Double** indique la valeur actuelle, ou valeur au jour d'aujourd'hui, d'une série de paiements ou de recettes futurs. Par exemple, lorsque vous empruntez de l'argent pour acheter une voiture, le montant du prêt correspond pour le prêteur à la valeur actuelle des remboursements mensuels que vous allez effectuer pour la voiture.

FV

Facultatif. **Double** indique la valeur future ou le solde en liquide souhaité au terme du dernier remboursement. Par exemple, la valeur future d'un emprunt est de 0 €, car il s'agit de sa valeur après le dernier remboursement. Toutefois, si vous souhaitez économiser 50 000 F sur 18 ans pour les études de votre enfant, ce montant constitue la valeur future. Si cet argument est omis, la position 0 est utilisée par défaut.

Due

Facultatif. Objet de type énumération DueDate indiquant la date d'échéance des paiements. Cet argument doit être `DueDate.EndOfPeriod` si les paiements sont dus à terme échu ou `DueDate.BegOfPeriod` si les paiements sont dus à terme à échoir. Si cet argument est omis, `DueDate.EndOfPeriod` est utilisé par défaut.

Guess

Facultatif. **Double** spécifiant la valeur que vous estimez est retourné par **Rate**. Si cet argument est omis, *Guess* a la valeur 0.1 (10 pour cent).

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*NPer* <= 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une annuité est une série de versements constants effectués sur une certaine période. Il peut s'agir d'un prêt (pour l'achat d'une maison, par exemple) ou d'un placement (tel qu'un plan d'épargne sur lequel de l'argent est versé chaque mois).

Pour tous les arguments, les sommes versées (telles que les dépôts sur un compte épargne) sont représentées par des

nombres négatifs ; les sommes reçues (telles que les distributions de dividendes) sont représentées par des nombres positifs.

Rate est calculé par itération. En commençant par la valeur de l'argument *Guess*, la fonction **Rate** répète le cycle de calcul jusqu'à ce que le résultat soit précis à 0,00001 pour cent près. La fonction **Rate** échoue si elle n'a pas pu trouver de résultat après 20 tentatives. Si votre estimation est de 10 pour cent et si la fonction **Rate** échoue, essayez d'attribuer une valeur différente à *Guess*.

Exemple

Cet exemple utilise la fonction **Rate** pour calculer le taux d'intérêt d'un emprunt, en fonction du nombre d'échéances (`TotPmts`), du montant des remboursements (`Payment`), de la valeur actuelle ou du capital de l'emprunt (`PVal`), de la valeur future de l'emprunt (`FVal`), d'un nombre indiquant si le paiement doit être effectué en début ou en fin de période (`PayType`), ainsi que de la valeur approximative du taux d'intérêt escompté (`Guess`).

VB

```
Sub TestRate()
    Dim PVal, Payment, TotPmts, APR As Double
    Dim PayType As DueDate

    ' Define percentage format.
    Dim Fmt As String = "##0.00"
    Dim Response As MsgBoxResult
    ' Usually 0 for a loan.
    Dim FVal As Double = 0
    ' Guess of 10 percent.
    Dim Guess As Double = 0.1
    PVal = CDbl(InputBox("How much did you borrow?"))
    Payment = CDbl(InputBox("What's your monthly payment?"))
    TotPmts = CDbl(InputBox("How many monthly payments do you have to make?"))
    Response = MsgBox("Do you make payments at the end of the month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    APR = (Rate(TotPmts, -Payment, PVal, FVal, PayType, Guess) * 12) * 100

    MsgBox("Your interest rate is " & Format(CInt(APR), Fmt) & " percent.")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des mots clés financiers

ArgumentException

Rename, fonction

Renomme un fichier sur disque ou un répertoire.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **Rename**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub Rename( _
   ByVal OldPath As String, _
   ByVal NewPath As String _
)
```

Paramètres

OldPath

Requis. Expression **String** qui spécifie le nom et l'emplacement du fichier existant. *OldPath* peut inclure le répertoire et le lecteur du fichier.

NewPath

Requis. Expression **String** qui spécifie le nouveau nom et le nouvel emplacement du fichier. *NewPath* peut inclure le répertoire et le lecteur de l'emplacement de destination. Le nom de fichier spécifié par *NewPath* ne peut pas déjà exister.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Le chemin d'accès n'est pas valide.
FileNotFoundException	53	Le fichier *OldPath* n'existe pas.
IOException	58	Le fichier *NewPath* existe déjà.
IOException	75	L'accès est non valide.
IOException	74	Impossible de renommer vers un autre périphérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Rename** renomme un fichier et le déplace vers un répertoire différent, si nécessaire. La fonction **Rename** peut déplacer un fichier dans des lecteurs, mais il peut uniquement renommer un répertoire existant lorsqu'à la fois *NewPath* et *OldPath* se trouvent sur le même lecteur. **Rename** ne peut pas créer de nouveau fichier ou répertoire.

L'utilisation de la fonction **Rename** sur un fichier ouvert génère une erreur. Vous devez fermer un fichier avant de le renommer. Les arguments **Rename** ne peuvent pas comprendre de caractères génériques à plusieurs caractères (*), ni à un caractère (?).

Remarque de sécurité
Lors de l'utilisation de **Rename** pour copier un fichier d'un emplacement non protégé vers un emplacement protégé, le fichier conserve les droits moins restreints. Assurez-vous que vous n'introduisez pas d'éventuel problème de sécurité.

Exemple

Cet exemple utilise la fonction **Rename** pour renommer un fichier. Pour cet exemple, supposons que les répertoires spécifiés existent déjà.

VB

```
Dim OldName, NewName As String
OldName = "OLDFILE"
' Define file names.
NewName = "NEWFILE"
' Rename file.
Rename(OldName, NewName)

OldName = "C:\OLDDIR\OLDFILE"
NewName = "C:\NEWDIR\NEWFILE"
' Move and rename file.
Rename(OldName, NewName)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : renommer un fichier dans Visual Basic

Comment : renommer un répertoire dans Visual Basic

Référence

Kill, fonction

ArgumentException Class

FileNotFoundException Class

IOException Class

Replace, fonction (Visual Basic)

Retourne une chaîne dans laquelle une sous-chaîne spécifiée a été remplacée par une autre sous-chaîne, un nombre de fois déterminé.

```
Public Function Replace(
   ByVal Expression As String,
   ByVal Find As String,
   ByVal Replacement As String,
   Optional ByVal Start As Integer = 1,
   Optional ByVal Count As Integer = -1,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
) As String
```

Paramètres

Expression

Requis. Expression de chaîne contenant la sous-chaîne à remplacer.

Find

Requis. Sous-chaîne à rechercher.

Replacement

Requis. Sous-chaîne de remplacement.

Start

Facultatif. Position dans *Expression* où la recherche de la sous-chaîne doit commencer. Si cet argument est omis, la position 1 est utilisée par défaut.

Count

Facultatif. Nombre de remplacements de la sous-chaîne à effectuer. Si cet argument est omis, la valeur par défaut est –1, ce qui implique l'exécution de tous les remplacements possibles.

Compare

Facultatif. Valeur numérique indiquant le type de comparaison à utiliser pour l'évaluation des sous-chaînes. Consultez la section Valeurs pour obtenir les valeurs.

Paramètres

L'argument *Compare* peut prendre les valeurs suivantes.

Constante	Description
Binary	Effectue une comparaison binaire.
Text	Effectue une comparaison de texte.

Valeur de retour

Replace retourne les valeurs suivantes.

Si	Valeur retournée par la fonction Replace
Find est de longueur nulle ou **Nothing**	Copie de *Expression*
Replace est de longueur nulle	Copie de *Expression* sans occurrence de *Find*

Expression est de longueur nulle ou **Nothing**, ou *Start* est supérieur à la longueur de *Expression*	**Nothing**
L'argument *Count* a la valeur 0	Copie de *Expression*

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Count* < -1 ou *Start* <= 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La valeur de retour de la fonction **Replace** est une chaîne qui commence à la position spécifiée par *Start* et qui se termine à la fin de la chaîne *Expression*, dont les substitutions sont effectuées comme spécifié par les valeurs *Find* et *Replace*.

Exemple

Cet exemple illustre la fonction **Replace**.

```
Dim TestString As String = "Shopping List"
' Returns "Shipping List".
Dim aString As String = Replace(TestString, "o", "i")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Filter, fonction (Visual Basic)
ArgumentException

Reset, fonction

Ferme tous les fichiers disque ouverts à l'aide de la fonction **FileOpen**.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **Reset**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub Reset()
```

Notes

La fonction **Reset** ferme tous les fichiers actifs ouverts par la fonction **FileOpen** et a la même fonction que **FileClose()** sans aucun paramètre.

Exemple

Cet exemple utilise la fonction **Reset** pour fermer tous les fichiers ouverts et écrire le contenu de leurs mémoires tampons sur le disque. Notez que la variable **Object** `FileNumber` est utilisée à la fois comme une chaîne et un nombre.

VB

```
' Open 5 files named TEST1, TEST2, etc.
Dim fileNumber As Integer
' Open 5 files.
For fileNumber = 1 To 5
   FileOpen(fileNumber, "TEST" & fileNumber, OpenMode.Output)
   PrintLine(fileNumber, "Hello World")
Next fileNumber
' Close files and write contents to disk.
Reset()
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileClose, fonction
End, instruction
FileOpen, fonction

RGB, fonction (Visual Basic)

Retourne une valeur **Integer** représentant une valeur de couleur RVB à partir d'un ensemble de composantes de couleur rouge, vert et bleu.

```
RGB( _
      Red As Integer, _
    Green As Integer, _
      Blue As Integer _
)
```

Paramètres

Red

Requis. **Integer** de 0 à 255 inclus, qui représente l'intensité de la composante rouge de la couleur.

Green

Requis. **Integer** de 0 à 255 inclus, qui représente l'intensité de la composante verte de la couleur.

Blue

Requis. **Integer** de 0 à 255 inclus, qui représente l'intensité de la composante bleue de la couleur.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Green*, *Blue*, ou *Red* en dehors de la plage qui s'étend de 0 à 255 inclus.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les propriétés et les méthodes de l'application qui acceptent une spécification de couleur s'attendent à la recevoir sous forme de nombre représentant une valeur de couleur RVB. Une valeur de couleur RVB spécifie l'intensité relative du rouge, du vert et du bleu composant ainsi la couleur spécifique à afficher.

Si la valeur d'un argument **RGB** est supérieure à 255, la valeur 255 est utilisée.

Le tableau suivant dresse la liste de quelques couleurs standard et des valeurs de rouge, vert et bleu qu'elles comprennent.

Couleur	Valeur Rouge	Valeur Vert	Valeur Bleu
Noir	0	0	0
Bleu	0	0	255
Vert	0	255	0
Cyan	0	255	255
Rouge	255	0	0
Magenta	255	0	255
Jaune	255	255	0
Blanc	255	255	255

Exemple

Cet exemple montre comment la fonction **RGB** est utilisée pour retourner un nombre entier représentant une valeur de couleur **RGB**.

VB

```
Dim red, rgbValue As Integer
Dim i As Integer = 75
' Return the value for red.
red = RGB(255, 0, 0)
' Same as RGB(75, 139, 203).
rgbValue = RGB(i, 64 + i, 128 + i)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

QBColor, fonction

ArgumentException Class

Right, fonction (Visual Basic)

Retourne une chaîne contenant un nombre spécifié de caractères depuis la partie droite d'une chaîne.

```
Public Shared Function Right( _
   ByVal str As String, _
   ByVal Length As Integer _
) As String
```

Paramètres

str

Requis. Expression **String** à partir de laquelle les caractères situés à l'extrême droite sont retournés.

Length

Requis. **Integer**. Expression numérique indiquant le nombre de caractères à retourner. Pour la valeur 0, une chaîne de longueur nulle ("") est retournée. Si le nombre de caractères est supérieur ou égal à celui de *str*, la chaîne entière est retournée.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Length* < 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Pour déterminer le nombre de caractères contenus dans *str*, utilisez la fonction **Len**. Si elle est utilisée dans un Windows Form ou dans toute autre classe qui a une propriété **Right** vous devez attribuer un nom complet à la fonction avec **Microsoft.VisualBasic.Right**.

Remarque

Dans les versions précédentes de Visual Basic, la fonction **RightB** retourne une chaîne en octets plutôt que des caractères. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Character Set). Toutes les chaînes Visual Basic en cours sont en Unicode et la fonction **RightB** n'est plus prise en charge.

Exemple

L'exemple suivant montre l'utilisation de la fonction **Right** pour retourner une sous-chaîne d'une expression **String** donnée. Dans une classe dotée d'une propriété **Right**, il peut être nécessaire d'attribuer un nom complet à la fonction **Right**.

VB

```
Dim TestString As String = "Hello World!"
' Returns "World!".
Dim subString As String = Microsoft.VisualBasic.Right(TestString, 6)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Left, fonction (Visual Basic)
Len, fonction (Visual Basic)
Mid, fonction (Visual Basic)
ArgumentException
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

RmDir, fonction

Supprime un répertoire existant.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **RmDir**. Pour plus d'informations, consultez My.Computer.FileSystem.DeleteDirectory, méthode.

```
Public Sub RmDir(ByVal Path As String)
```

Paramètres
Path

Requis. Expression **String** qui identifie le répertoire ou le dossier à supprimer. *Path* peut comprendre le lecteur. Si aucun lecteur n'est spécifié, **RmDir** supprime le répertoire du lecteur actif.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	52	*Path* n'est pas spécifié ou il est vide.
IOException	75	Le répertoire cible contient des fichiers.
FileNotFoundException	76	Le répertoire n'existe pas.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une erreur se produit si vous essayez d'utiliser **RmDir** sur un répertoire contenant des fichiers. Utilisez la fonction **Kill** pour supprimer tous les fichiers avant de tenter de supprimer un répertoire.

Exemple

Cet exemple utilise la fonction **RmDir** pour supprimer un répertoire existant.

VB

```
' Assume that MYDIR is an empty directory.
' Remove MYDIR.
RmDir("MYDIR")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Tâches
Comment : supprimer un répertoire dans Visual Basic
Référence
ChDir, fonction
CurDir, fonction
Kill, fonction

MkDir, fonction
ArgumentException Class
IOException Class
FileNotFoundException Class

Rnd, fonction (Visual Basic)

Retourne un nombre aléatoire de type **Single**.

```
Public Shared Function Rnd[(Number)] As Single
```

Paramètres

Number

Facultatif. Valeur **Single** ou toute expression **Single** valide.

Valeur de retour

Si Number est	La fonction Rnd génère
Inférieur à zéro	Le même nombre à chaque fois, en utilisant l'argument *Number* comme valeur initiale.
Supérieur à zéro	Le nombre aléatoire suivant dans la série.
Égal à zéro	Le dernier nombre généré.
Non spécifié	Le nombre aléatoire suivant dans la série.

Notes

La fonction **Rnd** retourne une valeur inférieure à 1, mais supérieure ou égale à zéro.

La valeur de *Number* détermine de quelle façon **Rnd** génère un nombre aléatoire :

Quelle que soit la valeur initiale donnée, la même séquence de nombres est générée, car chaque fois que la fonction **Rnd** est appelée, le nombre précédemment généré est utilisé comme valeur initiale pour le nombre suivant dans la séquence.

Avant d'appeler **Rnd**, utilisez l'instruction **Randomize** sans argument pour initialiser le générateur de nombres aléatoires à partir d'une valeur initiale basée sur l'horloge système.

Pour générer des entiers aléatoires dans une plage donnée, utilisez la formule suivante :

VB

```
randomvalue = CInt(Int((upperbound - lowerbound + 1) * Rnd() + lowerbound))
```

Ici, `upperbound` est le nombre le plus élevé de la plage et `lowerbound` est le nombre le plus bas de la plage.

Remarque

Pour répéter des séquences de nombres aléatoires, appelez **Rnd** avec un argument négatif juste avant d'utiliser **Randomize** avec un argument numérique. Si vous utilisez l'instruction **Randomize** en attribuant la même valeur à *Number*, la séquence précédente n'est pas répétée.

Remarque de sécurité

Étant donné que l'instruction **Random** et que la fonction **Rnd** commencent par une valeur initiale et génèrent des nombres c ompris dans une plage finie, les résultats peuvent être prévus par toute personne connaissant l'algorithme utilisé pour les cr éer. L'instruction **Random** et la fonction **Rnd** ne doivent donc pas être utilisées pour créer des nombres aléatoires qui seront utilisés en chiffrement.

Exemple

Cet exemple utilise la fonction **Rnd** pour générer une valeur entière aléatoire comprise entre 1 et 6.

VB

```
' Initialize the random-number generator.
```

```
Randomize()
' Generate random value between 1 and 6.
Dim value As Integer = CInt(Int((6 * Rnd()) + 1))
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : VBMath

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Randomize, fonction (Visual Basic)
Liste des mots clés mathématiques
Randomize, fonction (Visual Basic)

RSet, fonction

Retourne une chaîne alignée à droite contenant la chaîne spécifiée ajustée à la longueur spécifiée.

```
Public Shared Function RSet( _
       ByVal Source As String, _
       ByVal Length As Integer _
) As String
```

Paramètres

Source

Requis. Expression **String**. Chaîne à ajuster.

Length

Requis. Expression **Integer**. Longueur de chaîne retournée.

Notes

Si *Source* est plus long que *Length*, **RSet** retourne uniquement les caractères les plus à gauche, jusqu'à atteindre la longueur de *Length*. Si *Source* est plus court que *Length*, **RSet** ajoute des espaces à l'extrémité gauche de la chaîne afin d'obtenir la longueur appropriée.

Exemple

Cet exemple illustre l'utilisation de la fonction **RSet**.

VB

```
Dim TestString As String = "Right"
' Returns "      Right"
Dim rString As String = RSet(TestString, 11)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Liste des types de données (Visual Basic)
LSet, fonction

Concepts

Utilisation efficace des types de données

Fonctions Trim, LTrim et Rtrim

Retourne une chaîne contenant la copie d'une chaîne spécifiée sans espaces à gauche (**LTrim**), sans espaces à droite (**RTrim**) ou sans espaces à droite ni à gauche (**Trim**).

```
Public Shared Function LTrim(ByVal str As String) As String
Public Shared Function RTrim(ByVal str As String) As String
Public Shared Function Trim(ByVal str As String) As String
```

Paramètres

str

Requis. Toute expression **String** valide.

Notes

Les fonctions **LTrim**, **RTrim** et **Trim** suppriment les espaces à la fin des chaînes. Pour supprimer d'autres formes d'espaces blancs, comme les tabulations, utilisez la méthode System.String.Trim.

Exemple

L'exemple suivant utilise la fonction **LTrim** pour supprimer les espaces à gauche et la fonction **RTrim** pour supprimer les espaces à droite dans une variable chaîne. Il utilise la fonction **Trim** pour supprimer les deux types d'espaces.

VB

```
' Initializes string.
Dim TestString As String = "  <-Trim->  "
Dim TrimString As String
' Returns "<-Trim->  ".
TrimString = LTrim(TestString)
' Returns "  <-Trim->".
TrimString = RTrim(TestString)
' Returns "<-Trim->".
TrimString = LTrim(RTrim(TestString))
' Using the Trim function alone achieves the same result.
' Returns "<-Trim->".
TrimString = Trim(TestString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Left, fonction (Visual Basic)
Right, fonction (Visual Basic)
Trim

Fonctions S-Z

Le tableau suivant contient une liste des fonctions membres runtime Visual Basic.

SaveSetting	Second	Seek	SetAttr
Shell	SLN	Space	SPC
Split	Str	StrComp	StrConv
StrDup	StrReverse	Switch	SYD
SystemTypeName	TAB	TimeSerial	TimeValue
Trim	TypeName	UBound	UCase
Unlock	Val	VarType	VbTypeName
Weekday	WeekdayName	Write	WriteLine
Year			

Les fonctions mathématiques **Sgn**, **Sin**, **Sqr** et **Tan**, qui étaient disponibles en tant que fonctions membres runtime dans les versions 6.0 et antérieures de Visual Basic, ont été remplacées par des méthodes (Sign, Sin, Sqrt et Tan) de la classe Math de la bibliothèque de classes .NET Framework. Pour plus d'informations, consultez Fonctions mathématiques (Visual Basic).

Notes du développeur sur Smart Device

Les éléments suivants ne sont pas pris en charge dans les applications Smart Device : **SaveSetting**, **Seek**, **SetAttr**, **Shell**, **SPC**, **StrConv**, **TAB**, **Unlock**, **VarType**, **Write** et **WriteLine**.

Voir aussi

Référence

Fonctions A-C
Fonctions D-G
Fonctions H-L
Fonctions M-R

Autres ressources

Référence Visual Basic

SaveSetting, fonction

Enregistre ou crée une entrée d'application dans la base de registres de Windows.

La fonctionnalité **My** vous permet une meilleure productivité et de meilleures performances dans les opérations de Registre que **SaveSetting**. Pour plus d'informations, consultez My.Computer.Registry, objet.

```
Public Sub SaveSetting( _
   ByVal AppName As String, _
   ByVal Section As String, _
   ByVal Key As String, _
   ByVal Setting As String _
)
```

Paramètres

AppName

Requis. Expression **String** contenant le nom de l'application ou du projet auquel s'applique le paramètre.

Section

Requis. Expression **String** contenant le nom de la section dans laquelle le paramètre de clé est enregistré.

Key

Requis. Expression **String** contenant le nom du paramètre de clé enregistré.

Setting

Requis. Expression contenant la valeur définie pour *Key*.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	La clé du Registre n'a pas pu être créée ou l'utilisateur n'est pas connecté.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **SaveSetting** ajoute la clé à **HKEY_CURRENT_USER\Software\VB and VBA Program Settings**.

Si le paramètre de clé ne peut pas être enregistré pour une raison quelconque, une erreur se produit.

SaveSetting nécessite qu'un utilisateur soit connecté, car elle fonctionne sous la clé de Registre **HKEY_LOCAL_USER**, qui n'est active que lorsque l'utilisateur se connecte de façon interactive.

Les paramètres de Registre accessibles via un processus non interactif (tel que Mtx.exe) doivent être stockés au-dessous des clés de Registre **HKEY_LOCAL_MACHINE\Software** ou **HKEY_USER\DEFAULT\Software**.

SaveSetting requiert l'autorisation Registre **Write** et **Create** frlrfSystemSecurityPermissionsRegistryPermissionClassTopic.

Exemple

L'exemple suivant utilise d'abord la fonction **SaveSetting** pour entrer des données dans la base de registres de Windows pour l'application `MyApp`, puis utilise la fonction **DeleteSetting** pour les supprimer.

VB

```
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp","Startup", "Left", "50")
' Remove Startup section and all its settings from registry.
DeleteSetting ("MyApp", "Startup")
```

```
' Remove MyApp from the registry.
DeleteSetting ("MyApp")
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DeleteSetting, fonction
GetAllSettings, fonction
GetSetting, fonction
RegistryPermission Class

Concepts

Tâches courantes relatives au Registre

Second, fonction (Visual Basic)

Retourne une valeur de type **Integer** comprise entre 0 et 59 représentant la seconde de la minute.

```
Public Function Second(ByVal TimeValue As DateTime) As Integer
```

Paramètres
TimeValue

Requis. Valeur **Date** de laquelle vous souhaitez extraire la seconde.

Notes

Vous pouvez également obtenir la seconde de la minute en appelant **DatePart** et en spécifiant **DateInterval.Second** pour l'argument *Interval*.

Exemple

L'exemple suivant utilise la fonction **Second** pour obtenir la seconde la minute à partir d'une heure spécifiée. Dans l'environnement de développement, le littéral d'heure est affiché sous le format d'heure abrégée en utilisant les paramètres régionaux de votre code.

VB

```
Dim thisTime As Date
Dim thisSecond As Integer
thisTime = #4:35:17 PM#
thisSecond = Second(thisTime)
' thisSecond now contains 17.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Day, fonction (Visual Basic)
Hour, fonction (Visual Basic)
Minute, fonction
Now, propriété
TimeOfDay, propriété
DatePart, fonction (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

Seek, fonction

Retourne une valeur de type **Long** spécifiant la position de lecture/écriture actuelle dans un fichier ouvert à l'aide de la fonction **FileOpen** ou définit la position pour la prochaine opération de lecture/écriture dans un fichier ouvert à l'aide de la fonction **FileOpen**.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **Seek**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Overloads Function Seek( _
   ByVal FileNumber As Integer _
) As Long
' -or-
Public Overloads Sub Seek( _
   ByVal FileNumber As Integer, _
   ByVal Position As Long _
)
```

Paramètres

FileNumber

Requis. Valeur de type **Integer** contenant un numéro de fichier valide.

Position

Requis. Nombre compris entre 1 et 2 147 483 647, inclus, qui indique l'endroit où l'opération de lecture ou d'écriture suivante doit avoir lieu.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Seek** retourne une valeur comprise entre 1 et 2 147 483 647 (équivalent à 2^31 – 1), inclus.

Le tableau suivant décrit les valeurs de retour en fonction du mode d'accès au fichier :

Mode	Valeur de retour
Random	Numéro de l'enregistrement suivant lu ou écrit
Binary, **Input**, **Output**, **Append**	Position de l'octet à partir duquel l'opération suivante doit avoir lieu. Le premier octet d'un fichier occupe la position 1, le deuxième octet la position 2, et ainsi de suite.

Exemple

Cet exemple utilise la fonction **Seek** pour retourner la position actuelle du fichier. Nous supposons que `TestFile` est un fichier contenant les enregistrements de la structure `Record`.

VB

```
Structure Record   ' Define user-defined type.
   Dim ID As Integer
   Dim Name As String
End Structure
```

Dans le cas de fichiers ouverts en mode **Random**, la fonction **Seek** retourne le numéro de l'enregistrement suivant.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Random)
Do While Not EOF(1)
   WriteLine(1,Seek(1))    ' Write record number.
   FileGet(1, MyRecord, -1)    ' Read next record.
Loop
FileClose(1)
```

Dans le cas de fichiers ouverts en d'autres modes que le mode **Random**, la fonction **Seek** retourne la position de l'octet à partir duquel l'opération suivante doit s'effectuer. Nous supposons que `TestFile` est un fichier contenant quelques lignes de texte.

VB

```
' Report character position at beginning of each line.
Dim TextLine As String
FileOpen(1, "TESTFILE", OpenMode.Input)    ' Open file for reading.
While Not EOF(1)
' Read next line.
   TextLine = LineInput(1)
   ' Position of next line.
   MsgBox(Seek(1))
End While
FileClose(1)
```

Cet exemple utilise la fonction **Seek** pour définir la position de la prochaine opération de lecture ou d'écriture dans un fichier. Nous supposons que `People.txt` est un fichier contenant les enregistrements de la structure `Record`.

VB

```
Structure TestRecord
   Dim Name As String
   Dim ID As Integer
End Structure
```

Dans le cas de fichiers ouverts en d'autres modes que le mode **Random**, l'instruction **Seek** définit la position de l'octet à partir duquel l'opération suivante doit s'effectuer. Nous supposons que `TestFile` est un fichier contenant quelques lignes de texte.

VB

```
Dim someText As String = "This is a test string."
' Open file for output.
FileOpen(1, "TESTFILE", OpenMode.Input)
' Move to the third character.
Seek(1, 3)
Input(1, someText)
Console.WriteLine(someText)
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileGet, fonction

Loc, fonction

FileOpen, fonction

FilePut, fonction

IOException Class

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

SetAttr, fonction

Définit les informations relatives aux attributs d'un fichier.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S sur fichier que **SetAttr**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

```
Public Sub SetAttr( _
    ByVal PathName As String, _
    ByVal Attributes As FileAttribute _
)
```

Paramètres

PathName

Requis. Expression de type **String** spécifiant un nom de fichier. *PathName* peut comprendre un nom de répertoire ou de dossier, et un nom de lecteur.

Attributes

Requis. Expression constante ou numérique dont la somme spécifie des attributs de fichier.

Paramètres

Les valeurs d'énumération de l'argument *Attributes* sont les suivantes :

Valeur	Constante	Description
Normal	**vbNormal**	Normal (par défaut).
ReadOnly	**vbReadOnly**	Lecture seule.
Hidden	**vbHidden**	Caché.
System	**vbSystem**	Fichier système.
Volume	**vbVolume**	Étiquette de volume.
Directory	**vbDirectory**	Répertoire ou dossier.
Archive	**vbArchive**	Fichier modifié depuis la dernière sauvegarde.
Alias	**vbAlias**	Le fichier a un nom différent.

Remarque

Ces énumérations sont spécifiées par le langage Visual Basic. Les noms peuvent remplacer les valeurs réelles partout dans v otre code.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	52	*PathName* non valide ou inexistant.
ArgumentException	5	Le type *Attribute*' n'est pas valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Une erreur d'exécution se produit si vous essayez de définir les attributs d'un fichier ouvert.

Exemple

Cet exemple utilise la fonction **SetAttr** pour définir les attributs d'un fichier.

VB

```
' Set hidden attribute.
SetAttr("TESTFILE", vbHidden)
' Set hidden and read-only attributes.
SetAttr("TESTFILE", vbHidden Or vbReadOnly)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

FileAttr, fonction
GetAttr, fonction
ArgumentException Class
FileAttribute, énumération
My.Computer.FileSystem, objet

Shell, fonction

Exécute un programme exécutable et retourne un entier contenant l'ID du processus du programme s'il est toujours en cours d'exécution.

```
Public Function Shell( _
   ByVal PathName As String, _
   Optional ByVal Style As AppWinStyle = AppWinStyle.MinimizedFocus, _
   Optional ByVal Wait As Boolean = False, _
   Optional ByVal Timeout As Integer = -1 _
) As Integer
```

Paramètres

PathName

Requis. **String**.Nom du programme à exécuter avec n'importe quel argument requis et commutateur de ligne de commande. *PathName* peut également inclure le lecteur et le chemin d'accès ou au répertoire ou le dossier.

Si vous ne connaissez pas le chemin d'accès au programme, vous pouvez utiliser le My.Computer.FileSystem.GetFiles, méthode pour le rechercher. Par exemple, vous pouvez appeler `My.Computer.FileSystem.GetFiles("C:\", True, "testFile.txt")` qui retourne le chemin d'accès complet de chaque fichier nommé `testFile.txt` sur le lecteur C:\.

Style

Facultatif. **AppWinStyle**. Valeur choisie à partir du AppWinStyle, énumération spécifiant le style de la fenêtre dans laquelle le programme doit être exécuté. Si *Style* est omis, **Shell** utilise **AppWinStyle.MinimizedFocus** qui démarre le programme réduit et avec un focus.

Wait

Facultatif. **Boolean**. Valeur indiquant si la fonction **Shell** doit attendre la fin du programme. Si *Wait* est omis, **Shell** a la valeur **False**.

Timeout

Facultatif. **Integer**. Temps en millisecondes à attendre jusqu'à la fin si *Wait* a la valeur **True**. Si *Timeout* est omis, **Shell** utilise -1 ; il n'y a donc aucun délai et **Shell** ne retourne aucune valeur jusqu'à la fin du programme. Ainsi, si vous omettez *Timeout* ou lui affecter la valeur -1, il est possible que **Shell** ne retourne jamais le contrôle dans votre programme.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Style* se trouve en dehors de la plage de 0 à 9 compris.
FileNotFoundException	53	**Shell** ne peut pas trouver le fichier *PathName*.
NullReferenceException	91	*PathName* est **Nothing**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La valeur de retour de la fonction **Shell** varie selon que le programme nommé dans *PathName* est en cours d'exécution lorsque **Shell** retourne une valeur. Si vous affectez la valeur **True** à *Wait* et que le programme se termine avant l'expiration du délai, **Shell** retourne zéro. Si le délai expire, si vous omettez *Wait* ou lui affectez la valeur **False**, **Shell** retourne l'ID de processus du programme. L'ID du processus est un nombre unique qui identifie le programme en cours d'exécution.

Échec de démarrage

Si la fonction **Shell** ne peut pas démarrer le programme nommé, une erreur **FileNotFoundException** se produit. Par

exemple, cette erreur peut arriver lorsque vous essayez d'exécuter un programme 16 bits, tel que `command.com`, à partir d'une application à l'aide de System.Windows.Forms. Pour résoudre cette erreur, vous pouvez exécuter un programme 32 bits qui appelle le programme 16 bits souhaité. Dans le cas de `command.com`, vous pouvez également exécuter `cmd.exe`.

En attente de la conclusion

Par défaut, la fonction **Shell** exécute le programme de manière asynchrone. Cela signifie qu'un programme lancé par la fonction **Shell** peut ne pas s'exécuter complètement avant que les instructions suivant la fonction **Shell** ne soient exécutées. Si vous souhaitez attendre la fin du programme avant de continuer, vous devez affecter la valeur **True** à *Wait*.

Détermination du code de sortie

Un processus peut retourner un *code de sortie* lorsqu'il se termine. Toutefois, vous ne pouvez pas utiliser **Shell** pour récupérer ce code de sortie, parce que **Shell** retourne zéro s'il attend la conclusion et que le processus s'exécute dans un objet différent de **Shell**.

Pour récupérer le code de sortie d'un processus, vous devez écrire votre code utilisateur pour initialiser le processus et attendre la conclusion. L'exemple suivant montre comment initialiser un processus, attendre sa conclusion et récupérer son code de sortie.

```
Dim procID As Integer
Dim newProc As Diagnostics.Process
newProc = Diagnostics.Process.Start("C:\WINDOWS\NOTEPAD.EXE")
procID = newProc.Id
newProc.WaitForExit()
Dim procEC As Integer = -1
If newProc.HasExited Then
    procEC = newProc.ExitCode
End If
MsgBox("Process with ID " & CStr(ProcID) & _
    " terminated with exit code " & CStr(procEC))
```

Protection de la spécification de fichier

Vous devez toujours mettre l'intégralité du chemin d'accès et la spécification de fichiers entre guillemets, comme le montre l'exemple suivant :

```
ID = Shell("""C:\Program Files\display.exe"" -a -q", , True, 100000)
```

Chaque paire de guillemets doubles adjacents (**" "**) dans le littéral de chaîne est interprétée comme un caractère de guillemet double dans la chaîne. Ainsi, l'exemple précédent présente la chaîne suivante à la fonction **Shell** :

```
"C:\Program Files\display.exe" -a -q
```

Si le chemin d'accès n'est pas placé entre guillemets, Windows recherche un fichier appelé `Program.exe` dans le répertoire C:\, au lieu du fichier `display.exe` dans le répertoire C:\Program Files.

Remarque de sécurité

Si vous ne placez pas le chemin d'accès et la spécification de fichier entre guillemets, un problème de sécurité risque de se poser si le nom du fichier ou le nœud du chemin contient des espaces. Dans l'exemple précédent, le nœud de chemin `\Program Files` inclut un espace. Si la spécification n'avait pas été entre guillemets et qu'un programme appelé `Program.exe` avait été installé dans C:\ (par intrusion illicite par exemple), Windows l'aurait exécuté au lieu de `display.exe`.

Remarque de sécurité

La fonction **Shell** nécessite une permission de code non managée, qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

L'exemple suivant utilise la fonction **Shell** pour exécuter une application spécifiée par l'utilisateur. Si Microsoft.VisualBasic.AppWinStyle.NormalFocus est spécifié en tant que second argument, l'application est activée et affichée sous sa taille normale.

VB

```vb
Dim procID As Integer
' Run calculator.
procID = Shell("C:\Windows\system32\calc.exe", AppWinStyle.NormalFocus)
' The preceding path is for Windows XP.
' The Windows 2000 path is C:\WINNT\system32\calc.exe.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

AppWinStyle, énumération
My.Computer.FileSystem.GetFiles, méthode
AppActivate, fonction
ArgumentException
FileNotFoundException
NullReferenceException

SLN, fonction

Retourne une valeur de type **Double** indiquant l'amortissement linéaire d'un bien pour une période donnée.

```
Function SLN( _
   ByVal Cost As Double, _
   ByVal Salvage As Double, _
   ByVal Life As Double _
) As Double
```

Paramètres

Cost

Requis. Donnée de type **Double** indiquant le coût initial du bien.

Salvage

Requis. Donnée de type **Double** indiquant la valeur du bien à la fin de son cycle de vie.

Life

Requis. Donnée de type **Double** indiquant la durée du cycle de vie du bien.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Life* = 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La période d'amortissement doit être exprimée dans la même unité que l'argument *Life*. Tous les arguments doivent avoir pour valeur des nombres positifs.

Exemple

Cet exemple utilise la fonction **SLN** pour retourner l'amortissement linéaire d'un bien pour une période déterminée, en fonction du coût initial du bien (`InitCost`), de sa valeur résiduelle à l'issue de sa période d'utilisation (`SalvageVal`) et de sa durée de vie en nombre d'années (`LifeTime`).

VB

```
Dim InitCost, SalvageVal, LifeTime, DepYear As Double
Dim Fmt As String = "###,##0.00"

InitCost = CDbl(InputBox("What's the initial cost of the asset?"))
SalvageVal = CDbl(InputBox("Enter the asset's value at end of its life."))
LifeTime = CDbl(InputBox("What's the asset's useful life in years?"))

' Use the SLN function to calculate the deprecation per year.
Dim SlnDepr As Double = SLN(InitCost, SalvageVal, LifeTime)
Dim msg As String = "The depreciation per year: " & Format(SlnDepr, Fmt)
msg &= vbCrLf & "Year" & vbTab & "Linear" & vbTab & "Doubling" & vbCrLf

' Use the SYD and DDB functions to calculate the deprecation for each year.
For DepYear = 1 To LifeTime
    msg &= DepYear & vbTab & _
        Format(SYD(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbTab & _
        Format(DDB(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbCrLf
Next
MsgBox(msg)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

SYD, fonction

DDB, fonction

Liste des mots clés financiers

ArgumentException

Space, fonction (Visual Basic)

Retourne une chaîne composée d'un nombre spécifié d'espaces.

```
Public Shared Function Space(ByVal Number As Integer) As String
```

Paramètres
Number

Requis. Expression **Integer**. Nombre d'espaces souhaité dans la chaîne.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Number* < 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Space** est utile pour mettre en forme la sortie et effacer des données dans les chaînes de longueur fixe.

Exemple

L'exemple suivant utilise la fonction **Space** pour retourner une chaîne constituée d'un nombre spécifié d'espaces.

VB

```
Dim TestString As String
' Returns a string with 10 spaces.
TestString = Space(10)
' Inserts 10 spaces between two strings.
TestString = "Hello" & Space(10) & "World"
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Liste des manipulations de chaînes
SPC, fonction
ArgumentException

SPC, fonction

Utilisée avec la fonction **Print** ou **PrintLine** pour positionner la sortie.

```
Public Function SPC(ByVal Count As Short) As SPCInfo
```

Paramètres
Count

Requis. Nombre d'espaces à insérer avant d'afficher ou d'imprimer l'expression suivante dans une liste.

Notes

Si *Count* est inférieur à la largeur de la ligne de sortie, la position d'impression suivante se situe immédiatement après les espaces indiqués. Si *Count* est supérieur à la largeur de la ligne de sortie, la fonction **SPC** calcule la position d'impression suivante au moyen de la formule ci-après :

currentprintposition(+(*Count* **Mod** *width*))

Par exemple, si la position d'impression actuelle est 24, la largeur de la ligne de sortie est de 80 et si vous spécifiez **SPC(**90**)**, l'impression suivante commencera à la position 34 (position actuelle + le reste de 90/80). Si la différence entre la position d'impression actuelle et la largeur de la ligne de sortie est inférieure à *Count* (ou à *Count* **Mod** *width*), la fonction **SPC** passe au début de la ligne suivante et génère un nombre d'espaces égal à *Count* – (*width* – *currentprintposition*).

Remarque

Vérifiez que les colonnes tabulaires sont suffisamment larges pour contenir des lettres larges.

Exemple

Cet exemple utilise la fonction **SPC** pour positionner la sortie dans un fichier et dans la fenêtre **Sortie**.

VB

```
' The SPC function can be used with the Print function.
FileOpen(1, "TESTFILE", OpenMode.Output)   ' Open file for output.
Print(1, "10 spaces between here", SPC(10), "and here.")
FileClose(1)   ' Close file.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Mod, opérateur (Visual Basic)
Fonctions Print, PrintLine
Space, fonction (Visual Basic)
TAB, fonction
FileWidth, fonction

Split, fonction (Visual Basic)

Retourne un tableau à une dimension de base zéro et contenant le nombre spécifié de sous-chaînes.

```
Function Split(
   ByVal Expression As String,
   Optional ByVal Delimiter As String = " ",
   Optional ByVal Limit As Integer = -1,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
) As String()
```

Paramètres

Expression

Requis. Expression **String** contenant des sous-chaînes et des délimiteurs.

Delimiter

Facultatif. Tout caractère unique utilisé pour identifier les limites de sous-chaîne. Si *Delimiter* est omis, le caractère espace (" ") est utilisé comme délimiteur par défaut.

Limit

Facultatif. Nombre maximal des sous-chaînes dans lesquelles la chaîne d'entrée doit être fractionnée. La valeur par défaut, -1, indique que la chaîne d'entrée doit être fractionnée à chaque occurrence de la chaîne *Delimiter*.

Compare

Facultatif. Valeur numérique indiquant la comparaison à utiliser lors de l'évaluation des sous-chaînes. Consultez « Settings » pour obtenir les valeurs.

Retourne

Tableau de **String**. Si *Expression* est une chaîne de longueur nulle (""), **Split** retourne un tableau à un seul élément contenant une chaîne de longueur nulle. Si *Delimiter* est une chaîne de longueur nulle ou qu'il n'apparaît pas dans *Expression*, **Split** retourne un tableau à un seul élément contenant la chaîne *Expression* entière.

Paramètres

L'argument *Compare* peut prendre les valeurs suivantes.

Constante	Description	Valeur
CompareMethod.Binary	Effectue une comparaison binaire.	0
CompareMethod.Text	Effectue une comparaison de texte.	1

Notes

Par défaut, ou lorsque *Limit* est égal à -1, la fonction **Split** fractionne la chaîne d'entrée à chaque occurrence de la chaîne du délimiteur et retourne les sous-chaînes dans un tableau. Lorsque le paramètre *Limit* est supérieur à zéro, la fonction **Split** fractionne la chaîne aux premières occurrences *Limit* -1 du délimiteur et retourne un tableau avec les sous-chaînes résultant. Par exemple, `Split("a:b:c", ":")` retourne `{"a", "b", "c"}` du tableau pendant que `Split("a:b:c", ":", 2)` retourne `{"a", "b:c"}` du tableau.

Lorsque la fonction **Split** rencontre deux séparateurs dans une même ligne ou un séparateur au début ou à la fin de la chaîne, elle les interprète comme s'ils entouraient une chaîne vide (""). Par exemple, `Split("xx", "x")` retourne le tableau contenant trois chaînes vides : une située entre le début de la chaîne et le premier "x", une située entre les deux chaînes "x" et la dernière située entre le dernier "x" et la fin de la chaîne.

Ce tableau montre comment le *Delimiter* facultatif, les paramètres *Limit* et *Compare* peuvent modifier le comportement de la fonction **Split**.

Appel fractionné	Valeur de retour

`Split("42, 12, 19")`	{"42," , "12," , "19"}
`Split("42, 12, 19", ", ")`	{"42", "12", "19"}
`Split("42, 12, 19", ", ", 2)`	{"42", "12, 19"}
`Split("192.168.0.1", ".")`	{"192", "168", "0", "1"}
`Split("Alice and Bob", " AND ")`	{"Alice et Bob"}
`Split("Alice and Bob", " AND ", ,CompareMethod.Text)`	{"Alice", "Bob"}
`Split("someone@example.com", "@",1)`	{"xyz@example.com"}
`Split("someone@example.com", "@",2)`	{"xyz", "example.com"}

Exemple

L'exemple suivant montre comment fractionner une chaîne à ses espaces.

VB

```
Dim TestString As String = "Look at these!"
' Returns an array containing "Look", "at", and "these!".
Dim TestArray() As String = Split(TestString)
```

L'exemple suivant montre comment fractionner des chaînes disposant de plusieurs séparateurs dans une même ligne et éliminer les chaînes vides par filtrage.

VB

```
Dim TestString As String = "apple    pear banana  "
Dim TestArray() As String = Split(TestString)
' TestArray holds {"apple", "", "", "", "pear", "banana", "", ""}
Dim LastNonEmpty As Integer = -1
For i As Integer = 0 To TestArray.Length - 1
    If TestArray(i) <> "" Then
        LastNonEmpty += 1
        TestArray(LastNonEmpty) = TestArray(i)
    End If
Next
ReDim Preserve TestArray(LastNonEmpty)
' TestArray now holds {"apple", "pear", "banana"}
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes

Join, fonction (Visual Basic)

CompareMethod, énumération

Str, fonction

Retourne la représentation **String** d'un nombre.

```
Public Shared Function Str(ByVal Number As Object) As String
```

Paramètres

Number

Requis. Expression **Object** contenant toute expression numérique valide.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	5	*Number* n'est pas spécifié.
InvalidCastException	5	*Number* n'est pas un type numérique.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Lorsque des nombres sont convertis en chaînes, un espace à gauche est toujours réservé pour le signe de *Number*. Si *Number* est positif, la chaîne retournée contient un espace à gauche et le signe plus est implicite. Un nombre négatif inclura le signe moins (-) et pas d'espace à gauche.

Utilisez la fonction **Format** pour convertir les valeurs numériques que vous voulez mettre aux formats dates, heures ou monnaie ou dans d'autres formats définis par l'utilisateur. À la différence de la fonction **Str**, la fonction **Format** ne comprend pas d'espace à gauche pour le signe de *Number*.

Remarque

La fonction **Str** ne reconnaît que le point (**.**) comme séparateur décimal valide. Si d'autres séparateurs décimaux sont utilisés (dans les applications internationales, par exemple), convertissez le nombre en chaîne à l'aide de la fonction **CStr** ou **Format**. Pour obtenir la représentation sous forme de chaîne d'un nombre dans une culture particulière, utilisez la méthode `ToString(IFormatProvider)` du nombre. Par exemple, utilisez ToString lors de la conversion d'une valeur de type **Double** en une chaîne.

Exemple

L'exemple suivant utilise la fonction **Str** pour retourner la représentation **String** d'un nombre. Lorsqu'un nombre positif est converti en chaîne, un espace à gauche est toujours réservé à son signe.

VB

```
Dim TestString As String
' Returns " 459".
TestString = Str(459)
' Returns "-459.65".
TestString = Str(-459.65)
' Returns " 459.001".
TestString = Str(459.001)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des conversions
Format, fonction
Fonctions de conversion de types de données
Format, fonction
Val, fonction
ArgumentNullException
InvalidCastException

StrComp, fonction (Visual Basic)

Retourne -1, 0 ou 1, à partir du résultat d'une comparaison de chaînes.

```
Public Shared Function StrComp( _
   ByVal String1 As String, _
   ByVal String2 As String, _
   <Microsoft.VisualBasic.OptionCompareAttribute> _
   Optional ByVal Compare As Microsoft.VisualBasic.CompareMethod _
) As Integer
```

Paramètres

String1

Requis. Toute expression **String** valide.

String2

Requis. Toute expression **String** valide.

Compare

Facultatif. Spécifie le type de comparaison de chaînes. Si l'argument *Compare* est omis, le paramètre **Option Compare** détermine le type de comparaison.

Paramètres

L'argument *Compare* peut prendre les valeurs suivantes :

Constante	Description
Binary	Effectue une comparaison binaire à partir d'un ordre de tri dérivé des représentations binaires internes des caractères.
Text	Effectue une comparaison textuelle à partir d'un ordre de tri textuel ne tenant pas compte de la casse et déterminé par les informations actuelles de culture de votre application.

Valeur de retour

La fonction **StrComp** retourne les valeurs suivantes :

Si	La fonction StrComp retourne
String1 se situe avant *String2*	-1
String1 est égal à *String2*	0
String1 se situe après *String2*	1

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	La valeur *Compare* n'est pas valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les chaînes sont comparées par valeurs de tri alphanumérique en commençant par le premier caractère. Pour plus d'informations sur les comparaisons binaires, les comparaisons textuelles et l'ordre de tri, consultez Option Compare, instruction.

Remarque de sécurité

Si votre application prend des décisions relatives à la sécurité en fonction du résultat d'une comparaison ou d'une opération de modification des majuscules et des minuscules, l'opération doit utiliser la méthode System.String.Compare(System.String,System.String,System.StringComparison) et passer Ordinal ou OrdinalIgnoreCase pour l'argument *comparisonType*. Pour plus d'informations, consultez Comment les informations de culture affectent les chaînes dans Visual Basic.

Exemple

L'exemple suivant utilise la fonction **StrComp** pour retourner les résultats d'une comparaison de chaînes. Si le troisième argument est omis, le type de comparaison défini dans l'instruction **Option Compare** ou les paramètres par défaut du projet s'exécute alors.

VB

```
' Defines variables.
Dim TestStr1 As String = "ABCD"
Dim TestStr2 As String = "abcd"
Dim TestComp As Integer
' The two strings sort equally. Returns 0.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Text)
' TestStr1 sorts after TestStr2. Returns -1.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Binary)
' TestStr2 sorts before TestStr1. Returns 1.
TestComp = StrComp(TestStr2, TestStr1)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
InStr, fonction (Visual Basic)
ArgumentException

Autres ressources

Chaînes en Visual Basic
Introduction aux chaînes dans Visual Basic

StrConv, fonction

Retourne une chaîne convertie comme spécifié.

```
Public Shared Function StrConv( _
   ByVal str As String, _
   ByVal Conversion As Microsoft.VisualBasic.VbStrConv, _
   Optional ByVal LocaleID As Integer,
) As String
```

Paramètres

str

Requis. Expression **String** à convertir.

Conversion

Requis. Membre VbStrConv, énumération. Valeur d'énumération spécifiant le type de conversion à effectuer.

LocaleID

Facultatif. Valeur **LocaleID**, si différente de la valeur de la culture en cours de l'application. Pour plus d'informations sur les ID de paramètres régionaux et les informations de culture, consultez CultureInfo. (La valeur de la culture en cours de l'application est la valeur par défaut.)

Paramètres

L'argument *Conversion* peut prendre les valeurs suivantes :

Membre de l'énumération	Description
VbStrConv.None	N'effectue pas de conversion.
VbStrConv.LinguisticCasing	Utilise les règles linguistiques pour la casse au lieu du système de fichiers (par défaut). Valide seulement avec **VbStrConv.UpperCase** et **VbStrConv.LowerCase**.
VbStrConv.UpperCase	Convertit la chaîne en majuscules.
VbStrConv.LowerCase	Convertit la chaîne en minuscules.
VbStrConv.ProperCase	Convertit en majuscule la première lettre de chaque mot d'une chaîne.
VbStrConv.Wide *	Convertit les caractères étroits (demi-largeur) de la chaîne en caractères larges (pleine largeur).
VbStrConv.Narrow *	Convertit les caractères larges (pleine largeur) de la chaîne en caractères étroits (demi-largeur).
VbStrConv.Katakana **	Convertit les caractères Hiragana de la chaîne en caractères Katakana.
VbStrConv.Hiragana **	Convertit les caractères Katakana de la chaîne en caractères Hiragana.
VbStrConv.SimplifiedChinese *	Convertit les caractères chinois traditionnels en chinois simplifié.

VbStrConv.TraditionalChinese *	Convertit les caractères chinois simplifiés en chinois traditionnels.

* S'applique aux paramètres régionaux asiatiques.

** Ne s'applique qu'au Japon.

Remarque

Ces constantes sont spécifiées dans le Common Language Runtime du .NET Framework. En conséquence, elles peuvent être utilisées n'importe où dans votre code à la place des valeurs réelles. La plupart peuvent être combinées (par exemple, `UpperCase + Wide`), sauf lorsqu'elles s'excluent mutuellement (par exemple, `VbStrConv.Wide + VbStrConv.Narrow`).

Les séparateurs de mot valides pour la casse adéquate sont les suivants : Null (`Chr$(0)`), tabulation horizontale (`Chr$(9)`), saut de ligne (`Chr$(10)`), tabulation verticale (`Chr$(11)`), saut de page (`Chr$(12)`), retour chariot (`Chr$(13)`), espace (jeu de caractères codés sur un octet) (`Chr$(32)`). La valeur réelle pour un espace, qui peut être à demi-chasse/pleine chasse, varie en fonction du pays/de la région pour les cultures de l'est de l'Asie.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*LocaleID* n'est pas pris en charge, *Conversion* est inférieur à 0 ou supérieur à 2 048, ou la conversion n'est pas prise en charge pour les paramètres régionaux spécifiés.

Consultez la colonne "Numéro d'erreur" si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les constantes **VbStrConv.Wide**, **VbStrConv.Narrow**, **VbStrConv.SimplifiedChinese**, **VbStrConv.TraditionalChinese**, **VbStrConv.Katakana**et **VbStrConv.Hiragana** peuvent générer des erreurs d'exécution lorsqu'elles sont utilisées dans des paramètres régionaux où elles ne s'appliquent pas, mais pas toujours : les constantes **VbStrConv.Katakana** et **VbStrConv.Hiragana** peuvent être utilisées dans un système non japonais sur lequel le module de prise en charge du japonais est installé. En outre, l'utilisation des constantes **VbStrConv.Wide** et **VbStrConv.Narrow** est prise en charge sur tout système sur lequel une langue de l'est de l'Asie est installée.

Cette fonction utilise les informations relatives à la culture de l'application lors de la manipulation de la chaîne afin que les changements de casse soient appropriés pour les paramètres régionaux dans lesquels l'application est utilisée.

Remarque de sécurité

Si votre application prend des décisions relatives à la sécurité en fonction du résultat d'une comparaison ou d'une opération de modification des majuscules et des minuscules, l'opération doit utiliser la méthode System.String.Compare(System.String,System.String,System.StringComparison) et passer Ordinal ou OrdinalIgnoreCase pour l'argument *comparisonType*. Pour plus d'informations, consultez Comment les informations de culture affectent les chaînes dans Visual Basic.

Exemple

L'exemple suivant convertit entièrement le texte en lettres minuscules.

VB

```
Dim sText As String = "Hello World"
' Returns "hello world".
Dim sNewText As String = StrConv(sText, VbStrConv.LowerCase)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Fonctions Chr, ChrW
String, type de données (Visual Basic)
Fonctions de conversion de types de données
ArgumentException

StrDup, fonction

Retourne une chaîne ou un objet constitué du caractère spécifié répété le nombre de fois spécifié.

```
Public Shared Function StrDup( _
   ByVal Number As Integer, _
   ByVal Character As { Char | String } _
) As String
' -or-
Public Shared Function StrDup( _
   ByVal Number As Integer, _
   ByVal Character As Object _
) As Object
```

Paramètres

Number

Requis. Expression **Integer**. Longueur de la chaîne à retourner.

Character

Requis. Toute expression **Char**, **String** ou **Object** valide. Seul le premier caractère de l'expression sera utilisé. Si le caractère est de type **Object**, il doit contenir une valeur **Char** ou **String**

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Number* est inférieur à 0 ou le type *Character* n'est pas valide.
ArgumentNullException	5	*Character* a la valeur **Nothing**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Cette fonction retourne une expression **String** constituée de caractères répétés. Le caractère qui constitue la chaîne est le premier caractère de l'argument *Character* et il est dupliqué *Number* de fois.

Exemple

L'exemple suivant utilise la fonction **StrDup** pour retourner une chaîne de caractères dupliqués.

VB

```
Dim aString As String = "Wow! What a string!"
Dim aObject As New Object
Dim TestString As String
aObject = "This is a String contained within an Object"
' Returns "PPPPP"
TestString = StrDup(5, "P")
' Returns "WWWWWWWWWW"
TestString = StrDup(10, aString)
' Returns "TTTTTT"
TestString = CStr(StrDup(6, aObject))
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes

SPC, fonction

ArgumentException

ArgumentNullException

StrReverse, fonction (Visual Basic)

Retourne une chaîne dans laquelle l'ordre des caractères d'une chaîne donnée a été inversé.

```
Public Function StrReverse(ByVal Expression As String) As String
```

Paramètres
Expression

Requis. Expression de chaîne dont l'ordre des caractères doit être inversé. Si *Expression* est une chaîne de longueur nulle (""), la fonction retourne une chaîne de longueur nulle.

Notes

La fonction **StrReverse** retourne une chaîne qui contient les mêmes caractères que *Expression*, mais dans l'ordre opposé.

Exemple
VB

```
Dim TestString As String = "ABCDEFG"
' Returns "GFEDCBA".
Dim revString As String = StrReverse(TestString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Liste des manipulations de chaînes
InStrRev, fonction (Visual Basic)

Switch, fonction

Évalue une liste d'expressions et retourne une valeur **Object** correspondant à la première expression de la liste qui est **True**.

```
Public Function Switch( _
    ByVal ParamArray VarExpr() As Object _
) As Object
```

Paramètres

VarExpr

Requis. Tableau de paramètres **Object**. Doit comprendre un nombre pair d'éléments. Vous pouvez fournir une liste de variables ou d'expressions **Object** séparées par des virgules, ou un tableau unidimensionnel d'éléments **Object**.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Nombre impair d'arguments

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

L'argument fourni à *VarExpr* composé d'expressions et de valeurs fonctionnant par paires. La fonction **Switch** évalue les expressions impaires de l'index le plus bas à l'index le plus élevé dans *VarExpr*, et retourne la valeur paire associée à la première expression qui a la valeur **True**. Par exemple, si *VarExpr(0)* a la valeur **True**, **Switch** retourne *VarExpr(1)*, et si *VarExpr(0)* a la valeur **False**, mais *VarExpr(2)* est **True**, **Switch** retourne *VarExpr(3)*, et ainsi de suite.

Si vous ne fournissez pas l'argument *VarExpr*, **Switch** retourne **Nothing**.

Remarque

Les expressions dans la liste d'arguments peuvent inclure des appels de fonction. Pour préparer la liste d'arguments pour l'appel à **Switch**, le compilateur Visual Basic appelle chaque fonction de chaque expression. Ceci signifie qu'une fonction particulière sera appelée si une expression précédente de la liste d'arguments est **True**.

Exemple

L'exemple suivant utilise la fonction **Switch** pour retourner le nom de la langue correspondant au nom d'une ville. **Option Strict** doit être **Off**.

VB

```
Function matchLanguage(ByVal cityName As String) As String
    Return CStr(Microsoft.VisualBasic.Switch( _
        cityName = "London", "English", _
        cityName = "Rome", "Italian", _
        cityName = "Paris", "French"))
End Function
```

Comme l'espace de noms System.Diagnostics contient également une classe appelée Switch, un appel à la fonction **Switch** doit être qualifié avec l'espace de noms Microsoft.VisualBasic.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Choose, fonction

IIf, fonction

Select...Case, instruction (Visual Basic)

Concepts

Tableaux de paramètres

SYD, fonction

Retourne une valeur de type **Double** indiquant l'amortissement global d'un bien sur une période donnée.

```
Function SYD( _
   ByVal Cost As Double, _
   ByVal Salvage As Double, _
   ByVal Life As Double, _
   ByVal Period As Double _
) As Double
```

Paramètres

Cost

Requis. Donnée de type **Double** indiquant le coût initial du bien.

Salvage

Requis. Donnée de type **Double** indiquant la valeur du bien à la fin de son cycle de vie.

Life

Donnée de type **Double** indiquant la durée du cycle de vie du bien.

Period

Requis. Donnée de type **Double** indiquant la période sur laquelle l'amortissement du bien est calculé.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Salvage* < 0, *Period* > *Life*, ou *Period* <=0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les arguments *Life* et *Period* doivent être exprimés dans les mêmes unités. Par exemple, si *Life* est donné en mois, *Period* doit être également donné en mois. Tous les arguments doivent avoir pour valeur des nombres positifs.

Exemple

Cet exemple utilise la fonction **SYD** pour retourner l'amortissement linéaire d'un bien pour une période déterminée, en fonction du coût initial du bien (`InitCost`), de sa valeur résiduelle à l'issue de sa période d'utilisation (`SalvageVal`) et de sa durée de vie en nombre d'années (`LifeTime`).

VB

```
Dim InitCost, SalvageVal, LifeTime, DepYear As Double
Dim Fmt As String = "###,##0.00"

InitCost = CDbl(InputBox("What's the initial cost of the asset?"))
SalvageVal = CDbl(InputBox("Enter the asset's value at end of its life."))
LifeTime = CDbl(InputBox("What's the asset's useful life in years?"))

' Use the SLN function to calculate the deprecation per year.
Dim SlnDepr As Double = SLN(InitCost, SalvageVal, LifeTime)
Dim msg As String = "The depreciation per year: " & Format(SlnDepr, Fmt)
msg &= vbCrLf & "Year" & vbTab & "Linear" & vbTab & "Doubling" & vbCrLf

' Use the SYD and DDB functions to calculate the deprecation for each year.
For DepYear = 1 To LifeTime
    msg &= DepYear & vbTab & _
```

```
        Format(SYD(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbTab & _
        Format(DDB(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbCrLf
Next
MsgBox(msg)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Financial

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DDB, fonction
SLN, fonction
Liste des mots clés financiers
ArgumentException

SystemTypeName, fonction

Retourne une valeur **String** contenant le nom du type de données système d'une variable.

```
Public Function SystemTypeName(ByVal VbName As String) As String
```

Paramètres
VbName

Requis. Variable **String** contenant un nom de type Visual Basic.

Notes

SystemTypeName retourne le nom de type qualifié complet du Common Language Runtime (CLR) correspondant au nom de type Visual Basic. Par exemple, si *VbName* contient "Date", **SystemTypeName** retourne "System.DateTime". Si **SystemTypeName** ne reconnaît pas la valeur de l'argument *VbName*, elle retourne **Nothing** (pas la chaîne "Nothing").

Exemple

L'exemple suivant utilise la fonction **SystemTypeName** pour retourner les noms de types de données de plusieurs variables.

VB

```
Dim vbLongName As String = "Long"
Dim vbDateName As String = "Date"
Dim vbBadName As String = "Number"
Dim testSysName As String
testSysName = SystemTypeName(vbLongName)
' The preceding call returns "System.Int64".
testSysName = SystemTypeName(vbDateName)
' The preceding call returns "System.DateTime".
testSysName = SystemTypeName(vbBadName)
' The preceding call returns Nothing.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Liste des types de données (Visual Basic)
String, type de données (Visual Basic)
VbTypeName, fonction

TAB, fonction

Utilisée avec les fonctions **Print** ou **PrintLine** pour positionner la sortie.

```
Public Overloads Function TAB() As TABInfo
' -or-
Public Overloads Function TAB(ByVal Column As Short) As TABInfo
```

Paramètres
Column

Facultatif. Nombre de colonnes déplacées avant d'afficher ou d'imprimer l'expression suivante dans une liste. En cas d'omission, **TAB** déplace le point d'insertion au début de la zone d'impression suivante.

Notes

Si la position d'impression actuelle dans la ligne active est supérieure à *Column*, **TAB** passe à la valeur de colonne égale à *Column* sur la ligne de sortie suivante. Si *Column* est inférieure à 1, **TAB** déplace la position d'impression sur la colonne 1. Si *Column* est supérieure à la largeur de la ligne de sortie, **TAB** calcule la prochaine position d'impression à l'aide de la formule :

Colonne Mod Largeur

Par exemple, si *width* a la valeur 80 et si vous spécifiez **TAB(**90**)**, l'impression suivante commence à la colonne 10 (le reste de 90/80). Si *Column* est inférieure à la position d'impression actuelle, l'impression commence à la ligne suivante, au niveau de la position d'impression calculée. Si la position d'impression calculée est supérieure à la position d'impression actuelle, la sortie commence à la même ligne, au niveau de la position calculée.

La position d'impression de l'extrême gauche sur une ligne de sortie est toujours 1. Si vous utilisez les fonctions **Print** ou **PrintLine** pour imprimer dans des fichiers, la position d'impression de l'extrême droite correspond à la largeur actuelle du fichier de sortie, laquelle peut être définie à l'aide de la fonction **FileWidth**.

La fonction **TAB** peut également être utilisée avec la fonction **WriteLine**. Elle ne peut pas être utilisée avec System.Diagnostics.Debug.WriteLine ou System.Console.WriteLine.

Remarque

Vérifiez que les colonnes tabulaires sont suffisamment larges pour contenir des lettres larges.

Exemple

Cet exemple utilise la fonction **TAB** pour positionner la sortie dans un fichier et dans la fenêtre **Sortie**.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output) ' Open file for output.
' The second word prints at column 20.
Print(1, "Hello", TAB(20), "World.")
' If the argument is omitted, cursor is moved to the next print zone.
Print(1, "Hello", TAB(), "World")
FileClose(1)
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence

Mod, opérateur (Visual Basic)
Fonctions Print, PrintLine
Space, fonction (Visual Basic)
SPC, fonction
FileWidth, fonction

TimeSerial, fonction (Visual Basic)

Retourne une valeur **Date** représentant une heure, une minute et une seconde spécifiées, l'indication de date ayant la valeur 1er janvier de l'année 1.

```
Public Function TimeSerial( _
   ByVal Hour As Integer, _
   ByVal Minute As Integer, _
   ByVal Second As Integer _
) As DateTime
```

Paramètres

Hour

Requis. Expression **Integer** comprise entre 0 et 23. Cependant, les valeurs non comprises dans cette plage sont également acceptées.

Minute

Requis. Expression **Integer** comprise entre 0 et 59. Cependant, les valeurs non comprises dans cette plage sont également acceptées. La valeur de *Minute* est ajoutée à l'heure calculée, et donc une valeur négative spécifie les minutes précédant cette heure.

Second

Requis. Expression **Integer** comprise entre 0 et 59. Cependant, les valeurs non comprises dans cette plage sont également acceptées. La valeur de *Second* est ajoutée à la seconde calculée, et donc une valeur négative spécifie les secondes précédant cette minute.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	L'argument se trouve en dehors de la plage s'étendant de -2 147 483 648 à 2 147 483 647.
ArgumentOutOfRangeException	9	L'heure calculée est inférieure à -24 heures.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

L'exemple à suivre contient des valeurs d'arguments négatives, nulles et positives. La fonction **TimeSerial** retourne une période représentant 15 minutes avant trois heures avant midi, c'est-à-dire 8:45:00.

```
Dim alarmTime As Date = TimeSerial(12 - 3, -15, 0)
```

Si *Minute* ou *Second* dépasse la plage normale, la valeur est appliquée à l'unité supérieure pertinente. Par exemple, si vous spécifiez 75 minutes, ce chiffre est interprété comme une heure et 15 minutes.

La fonction **TimeSerial** réduit le total des secondes avec un modulo 86 400, qui correspond au nombre de secondes que contient un jour. Ainsi, l'heure retournée se situe toujours entre 00:00:00 et 23:59:59.

Le type de données **Date** inclut les composants de date. **TimeSerial** affecte la valeur 1 à tous ces composants, ainsi la valeur retournée représente le premier jour de l'année 1. Toutefois, si les valeurs des arguments entraînent le temps calculé à dépasser 24 heures, le jour est incrémenté comme il convient. Dans l'exemple suivant, les valeurs de *Hour* et *Minute* ont comme résultat une heure combinée supérieure à 24 heures.

```
MsgBox(TimeSerial(23, 75, 0))
```

```
' The preceding statement displays "1/2/0001 12:15:00 AM".
```

Si les valeurs des arguments ont pour résultat un temps calculé négatif, les informations de date ont la valeur 1/1/0001 et les informations d'heure sont ajustées entre 00:00:00 et 23:59:59. Toutefois, si le temps calculé en heures est inférieur à -24, une erreur **ArgumentOutOfRangeException** s'est produite.

Étant donné que chaque valeur **Date** est prise en charge par une structure System.DateTime, ses méthodes vous donnent des possibilités supplémentaires pour assembler une valeur **Date**. Par exemple, vous pouvez utiliser l'un des constructeurs DateTime surchargés pour remplir une variable **Date** à l'aide de la combinaison de composants souhaitée. L'exemple suivant affecte à `newDateTime` la valeur du 6 mai 1978 à un dixième de seconde avant 8 heures 30 :

```
Dim newDateTime As Date = New Date(1978, 5, 6, 8, 29, 59, 900)
```

Exemple

L'exemple suivant utilise la fonction **TimeSerial** pour retourner un temps pour l'heure, la minute et la seconde spécifiées.

VB

```
Dim thisTime As Date
thisTime = TimeSerial(16, 35, 17)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

DateSerial, fonction (Visual Basic)
DateValue, fonction (Visual Basic)
Hour, fonction (Visual Basic)
Minute, fonction
Now, propriété
Second, fonction (Visual Basic)
TimeValue, fonction (Visual Basic)

TimeValue, fonction (Visual Basic)

Retourne une valeur **Date** contenant l'indication d'heure représentée par une chaîne, l'indication de date ayant la valeur 1er janvier de l'année 1.

```
Public Function TimeValue(ByVal StringTime As String) As DateTime
```

Paramètres
StringTime

Requis. Expression **String** représentant une valeur de date/d'heure située entre 00:00:00 le 1er janvier de l'année 1 et 23:59:59 le 31 décembre 9999.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	13	*StringTime* inclut des informations de date non valides.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Vous pouvez entrer des heures valides en utilisant une horloge au format 12 ou 24 heures. Par exemple, `"2:24 P.M."` et `"14:24"` sont deux arguments *StringTime* valides.

Si l'argument *StringTime* comprend une indication de date, la fonction **TimeValue** ne l'inclut pas dans la valeur retournée. Toutefois, si *StringTime* inclut des informations de date non valides, par exemple "32 janvier", une erreur **InvalidCastException** se produit.

Exemple

L'exemple suivant utilise la fonction **TimeValue** pour convertir une chaîne en heure. Vous pouvez également utiliser des littéraux de date pour assigner directement une heure à une variable **Date**.

VB

```
Dim thisTime As Date
thisTime = TimeValue("4:35:17 PM")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
DateSerial, fonction (Visual Basic)
DateValue, fonction (Visual Basic)
Hour, fonction (Visual Basic)
Minute, fonction
Now, propriété
Second, fonction (Visual Basic)
TimeSerial, fonction (Visual Basic)
DateTime

Fonctions Trim, LTrim et Rtrim

Retourne une chaîne contenant la copie d'une chaîne spécifiée sans espaces à gauche (**LTrim**), sans espaces à droite (**RTrim**) ou sans espaces à droite ni à gauche (**Trim**).

```
Public Shared Function LTrim(ByVal str As String) As String
Public Shared Function RTrim(ByVal str As String) As String
Public Shared Function Trim(ByVal str As String) As String
```

Paramètres

str

Requis. Toute expression **String** valide.

Notes

Les fonctions **LTrim**, **RTrim** et **Trim** suppriment les espaces à la fin des chaînes. Pour supprimer d'autres formes d'espaces blancs, comme les tabulations, utilisez la méthode System.String.Trim.

Exemple

L'exemple suivant utilise la fonction **LTrim** pour supprimer les espaces à gauche et la fonction **RTrim** pour supprimer les espaces à droite dans une variable chaîne. Il utilise la fonction **Trim** pour supprimer les deux types d'espaces.

VB

```
' Initializes string.
Dim TestString As String = "  <-Trim->  "
Dim TrimString As String
' Returns "<-Trim->  ".
TrimString = LTrim(TestString)
' Returns "  <-Trim->".
TrimString = RTrim(TestString)
' Returns "<-Trim->".
TrimString = LTrim(RTrim(TestString))
' Using the Trim function alone achieves the same result.
' Returns "<-Trim->".
TrimString = Trim(TestString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des manipulations de chaînes
Left, fonction (Visual Basic)
Right, fonction (Visual Basic)
Trim

TypeName, fonction (Visual Basic)

Retourne une valeur **String** contenant des informations sur le type de données d'une variable.

```
Public Function TypeName(ByVal VarName As Object) As String
```

Paramètres

VarName

Requis. Variable **Object**. Si **Option Strict** est **Off**, vous pouvez passer une variable de n'importe quel type de données à l'exception d'une structure.

Notes

Le tableau suivant illustre les valeurs **String** retournées par **TypeName** pour différents contenus de *VarName*.

Contenu de *VarName*.	**Chaîne retournée**
Type valeur **True** ou **False** 16 bits	"Boolean"
Type valeur binaire 8 bits	"Byte"
Type valeur de caractère 16 bits	"Char"
Type valeur de date et heure 64 bits	"Date"
Type référence indiquant des données manquantes ou inexistantes	"DBNull"
Type valeur numérique à virgule fixe 128 bits	"Decimal"
Type valeur numérique à virgule flottante 64 bits	"Double"
Type valeur entier 32 bits	"Integer"
Type référence pointant vers un objet non spécialisé	"Object"
Type référence pointant vers un objet spécialisé créé à partir de la classe *objectclass*	"*objectclass*"
Type valeur entier 64 bits	"Long"
Type référence auquel aucun objet n'est actuellement assigné	"Nothing"
Type valeur entier 8 bits signé	"SByte"
Type valeur entier 16 bits	"Short"
Type valeur numérique à virgule flottante 32 bits	"Single"
Type référence pointant vers une chaîne de caractères 16 bits	"String"
Type valeur entier 32 bits non signé	"UInteger"
Type valeur entier 64 bits non signé	"ULong"

Type valeur entier 16 bits non signé	"UShort"

Si *VarName* est un tableau, la chaîne retournée peut être l'une des chaînes du tableau précédent suivie de parenthèses vides. Par exemple, si *VarName* pointe vers un tableau d'entiers, **TypeName** retourne "Integer()".

Lorsque **TypeName** retourne le nom d'un type référence tel qu'une classe, la fonction retourne uniquement le nom lui-même et pas le nom qualifié. Par exemple, si *VarName* pointe vers un objet de classe System.Drawing.Printing.PaperSource, **TypeName** retourne "PaperSource". Notez que si la variable est déclarée comme étant d'un certain type de classe, mais qu'aucun objet ne lui a été assigné, **TypeName** retourne "Nothing".

Exemple

L'exemple suivant utilise la fonction **TypeName** pour retourner des informations sur les types de données concernant plusieurs variables.

VB

```
Dim testType As String
Dim strVar As String = "String for testing"
Dim decVar As Decimal
Dim intVar, arrayVar(5) As Integer
testType = TypeName(strVar)
' The preceding call returns "String".
testType = TypeName(decVar)
' The preceding call returns "Decimal".
testType = TypeName(intVar)
' The preceding call returns "Integer".
testType = TypeName(arrayVar)
' The preceding call returns "Integer()".
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des types de données (Visual Basic)
IsArray, fonction (Visual Basic)
IsDate, fonction (Visual Basic)
IsDBNull, fonction
IsError, fonction
IsNothing, fonction
IsNumeric, fonction (Visual Basic)
IsReference, fonction

UBound, fonction (Visual Basic)

Retourne le plus grand indice disponible pour la dimension indiquée d'un tableau.

```
Public Function UBound( _
   ByVal Array As System.Array, _
   Optional ByVal Rank As Integer = 1 _
) As Integer
```

Paramètres

Array

Requis. Tableau de tout type de données. Tableau dans lequel vous voulez trouver le plus grand indice possible d'une dimension.

Rank

Facultatif. **Integer**. Dimension pour laquelle le plus grand indice possible doit être retourné. Utilisez la valeur 1 pour la première dimension, la valeur 2 pour la deuxième, etc. Si l'argument *Rank* est omis, 1 est supposé.

Valeur de retour

Integer. Valeur la plus élevée que l'indice pour la dimension spécifiée peut contenir. Si *Array* a un seul élément, **UBound** retourne 0. Si *Array* n'a pas d'éléments, par exemple, si c'est une chaîne de longueur nulle, **UBound** retourne -1.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentNullException	9	*Array* est **Nothing**
RankException	9	*Rank* est inférieur à 1 ou *Rank* est supérieur au rang de *Array*.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Compte tenu que les indices de tableau commencent à 0, la longueur d'une dimension est supérieure de un à l'indice le plus grand disponible pour cette dimension.

Pour un tableau avec les dimensions suivantes, **UBound** retourne les valeurs dans le tableau suivant :

```
Dim a(100, 5, 4) As Byte
```

Appel à UBound	Valeur de retour
`UBound(a, 1)`	100
`UBound(a, 2)`	5
`UBound(a, 3)`	4

Vous pouvez utiliser **UBound** pour déterminer le nombre total d'éléments dans un tableau, mais vous devez ajuster la valeur de retour pour prendre en compte le fait que les indices commencent à 0. L'exemple suivant calcule la taille totale du tableau `a` de l'exemple précédent :

```
Dim total As Integer
total = (UBound(A, 1) + 1) * (UBound(A, 2) + 1) * (UBound(A, 3) + 1)
```

La valeur calculée pour `total` est 3 030, c'est-à-dire 101 * 6 * 5.

Exemple

Cet exemple utilise la fonction **UBound** pour déterminer le plus grand indice disponible pour la dimension indiquée d'un tableau.

VB

```
Dim highest, bigArray(10, 15, 20), littleArray(6) As Integer
highest = UBound(bigArray, 1)
highest = UBound(bigArray, 3)
highest = UBound(littleArray)
' The three calls to UBound return 10, 20, and 6 respectively.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

LBound, fonction (Visual Basic)
Dim, instruction (Visual Basic)
ReDim, instruction (Visual Basic)
ArgumentException
RankException

UCase, fonction (Visual Basic)

Retourne une chaîne ou un caractère contenant la chaîne spécifiée convertie en majuscules.

```
Public Shared Function UCase(ByVal Value As Char) As Char
' -or-
Public Shared Function UCase(ByVal Value As String) As String
```

Paramètres
Value

Requis. Toute expression **String** ou **Char** valide.

Notes

Seules les lettres minuscules sont converties en majuscules ; toutes les majuscules et les caractères autres que des lettres demeurent inchangés.

Cette fonction utilise les informations relatives à la culture de l'application lors de la manipulation de la chaîne afin que les changements de casse soient appropriés pour les paramètres régionaux dans lesquels l'application est utilisée.

Remarque de sécurité

Si votre application prend des décisions relatives à la sécurité en fonction du résultat d'une comparaison ou d'une opération de modification des majuscules et des minuscules, l'opération doit utiliser la méthode System.String.Compare(System.String,System.String,System.StringComparison) et passer Ordinal ou OrdinalIgnoreCase pour l'argument *comparisonType*. Pour plus d'informations, consultez Comment les informations de culture affectent les chaînes dans Visual Basic.

Exemple

L'exemple suivant utilise la fonction **UCase** pour retourner une version en majuscules d'une chaîne.

VB

```
' String to convert.
Dim LowerCase As String = "Hello World 1234"
' Returns "HELLO WORLD 1234".
Dim UpperCase As String = UCase(LowerCase)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Liste des manipulations de chaînes
LCase, fonction (Visual Basic)
Autres ressources
Chaînes en Visual Basic
Introduction aux chaînes dans Visual Basic

Val, fonction

Retourne le nombre contenu dans une chaîne de caractère sous la forme d'une valeur numérique d'un type approprié.

```
Public Overloads Function Val(ByVal InputStr As String) As Double
' -or-
Public Overloads Function Val(ByVal Expression As Object) As Double
' -or-
Public Overloads Function Val(ByVal Expression As Char) As Integer
```

Paramètres

Expression, *InputStr*

Requis. Toute expression **String**, variable **Object** ou valeur **Char** valide. Si *Expression* est de type **Object**, sa valeur doit pouvoir être convertie en **String** ; sinon, une erreur ArgumentException se produit.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
OverflowException	6	*InputStr* est trop grand.
InvalidCastException	13	Le nombre est incorrect.
ArgumentException	438	L'expression de type **Object** ne peut pas être convertie en **String**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La fonction **Val** arrête la lecture de la chaîne au premier caractère ne faisant apparemment pas partie d'un nombre. Les symboles et caractères fréquemment associés aux valeurs numériques, comme le signe $ et les virgules, ne sont pas reconnus. En revanche, la fonction reconnaît les préfixes `&O` (pour octal) et `&H` (pour hexadécimal). Les espaces, les tabulations et les caractères de saut de ligne ne sont pas pris en compte dans l'argument.

L'appel suivant retourne la valeur 1615198 :

```
Val(" 1615 198th Street N.E.")
```

L'appel suivant retourne la valeur décimale -1.

```
Val("&HFFFF")
```

Remarque

La fonction **Val** ne reconnaît que le point (**.**) comme séparateur décimal valide. Lors de l'utilisation d'autres séparateurs déci maux (applications internationales, par exemple), convertissez la chaîne en nombre à l'aide de la fonction **CDbl** ou **CInt**. Pour convertir la représentation sous forme de chaîne d'un nombre utilisé dans une culture particulière en une valeur numérique, utilisez la méthode `Parse(String, IFormatProvider)` de type numérique. Par exemple, utilisez System.Double.Parse(System.String,System.IFormatProvider) lors de la conversion d'une chaîne en **Double**.

Exemple

L'exemple suivant utilise la fonction **Val** pour retourner les nombres contenus dans chaque chaîne. **Val** arrête la conversion au premier caractère impossible à interpréter comme un chiffre, un modificateur numérique, une ponctuation numérique ou un espace blanc.

VB

```vb
Dim valResult As Double
' The following line of code sets valResult to 2457.
valResult = Val("2457")
' The following line of code sets valResult to 2457.
valResult = Val(" 2 45 7")
' The following line of code sets valResult to 24.
valResult = Val("24 and 57")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Conversion

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Str, fonction
Fonctions de conversion de types de données
OverflowException
InvalidCastException
ArgumentException

VarType, fonction (Visual Basic)

Retourne une valeur **Integer** contenant la classification de type de données d'une variable.

```
Public Function VarType(ByVal VarName As Object) As VariantType
```

Paramètres

VarName

Requis. Variable **Object**. Si **Option Strict** est **Off**, vous pouvez passer une variable de n'importe quel type de données à l'exception d'une structure.

Notes

La valeur entière retournée par **VarType** est un membre de l'énumération VariantType.

Le tableau suivant illustre les valeurs retournées par **VarType** pour les cas spéciaux de *VarName*.

Type de données représenté par *VarName*	**Valeur retournée par VarType**
Nothing (Visual Basic)	**VariantType.Object**
DBNull	**VariantType.Null**
Énumération	Type de données sous-jacent (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**)
Tableau	Bits OR de type d'élément de tableau et **VariantType.Array**
Tableaux de tableaux	Bits OR de **VariantType.Object** et **VariantType.Array**
Structure (System.ValueType)	**VariantType.UserDefinedType**
Exception	**VariantType.Error**
Inconnu	**VariantType.Object**

Exemple

L'exemple suivant utilise la fonction **VarType** pour retourner des informations sur la classification des types de données concernant plusieurs variables.

VB

```
Dim testString As String = "String for testing"
Dim testObject As New Object
Dim testNumber, testArray(5) As Integer
Dim testVarType As VariantType
testVarType = VarType(testVarType)
' Returns VariantType.Integer.
testVarType = VarType(testString)
' Returns VariantType.String.
testVarType = VarType(testObject)
' Returns VariantType.Object.
testVarType = VarType(testNumber)
' Returns VariantType.Integer.
testVarType = VarType(testArray)
' Returns the bitwise OR of VariantType.Array and VariantType.Integer.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Liste des types de données (Visual Basic)
Object, type de données
VariantType, énumération

VbTypeName, fonction

Retourne une valeur **String** contenant le nom du type de données Visual Basic d'une variable.

```
Public Function VbTypeName(ByVal UrtName As String) As String
```

Paramètres
UrtName

Requis. Variable **String** contenant un nom de type utilisé par le Common Language Runtime.

Notes

VbTypeName retourne le nom du type Visual Basic correspondant au nom de type du Common Language Runtime. Par exemple, si *UrtName* contient "Int32" ou "System.Int32", **VbTypeName** retourne "Integer". Si **VbTypeName** ne reconnaît pas la valeur de l'argument *UrtName*, elle retourne **Nothing** (pas la chaîne "Nothing").

Le nom de type dans *UrtName* peut être obtenu à partir de diverses sources, par exemple à partir de la propriété Name dans le tableau MemberInfo de la classe Type.

Exemple

L'exemple suivant utilise la fonction **VbTypeName** pour retourner les noms de types de données de plusieurs variables.

VB

```
Dim sysDateName As String = "System.DateTime"
Dim sysShortName As String = "Int16"
Dim sysBadName As String = "Nonsense"
Dim testVbName As String
testVbName = VbTypeName(sysDateName)
' Returns "Date".
testVbName = VbTypeName(sysShortName)
' Returns "Short".
testVbName = VbTypeName(sysBadName)
' Returns Nothing.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Information

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Liste des types de données (Visual Basic)
String, type de données (Visual Basic)
SystemTypeName, fonction

Weekday, fonction (Visual Basic)

Retourne une valeur de type **Integer** représentant le jour de la semaine.

```
Public Function Weekday( _
   ByVal DateValue As DateTime, _
   Optional ByVal DayOfWeek As FirstDayOfWeek = FirstDayOfWeek.Sunday _
) As Integer
```

Paramètres

DateValue

Requis. Valeur **Date** pour laquelle vous souhaitez déterminer le jour de la semaine.

DayOfWeek

Facultatif. Valeur choisie dans l'énumération **FirstDayOfWeek** qui spécifie le premier jour de la semaine. S'il n'est pas spécifié, **FirstDayOfWeek.Sunday** est utilisé.

Paramètres

L'argument *DayOfWeek* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur	Description
FirstDayOfWeek.System	0	Premier jour de la semaine spécifiée dans les paramètres système
FirstDayOfWeek.Sunday	1	Dimanche (valeur par défaut)
FirstDayOfWeek.Monday	2	Lundi (conformément à la norme ISO 8601, section 3.17)
FirstDayOfWeek.Tuesday	3	Mardi
FirstDayOfWeek.Wednesday	4	Mercredi
FirstDayOfWeek.Thursday	5	Jeudi
FirstDayOfWeek.Friday	6	Vendredi
FirstDayOfWeek.Saturday	7	Samedi

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*DayOfWeek* est inférieur à 0 ou supérieur à 7.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La valeur retournée par la fonction **Weekday** est le jour de la semaine auquel *DateValue* correspond, par rapport au premier jour de la semaine défini par la valeur *DayOfWeek*. Par exemple, si la date fait référence à un mercredi et que le lundi est défini comme premier jour de la semaine, la fonction retourne la valeur 3.

Remarque

Weekday utilise le paramètre de calendrier actuel de la propriété CurrentCulture de la classe CultureInfo dans l'espace de noms System.Globalization. Les valeurs par défaut de **CurrentCulture** dépendent des paramètres du **Panneau de configuration**.

Exemple

L'exemple suivant utilise la fonction **Weekday** pour obtenir le jour de la semaine à partir d'une date spécifiée.

VB

```
Dim oldDate As Date
Dim oldWeekDay As Integer
oldDate = #2/12/1969#
oldWeekDay = Weekday(oldDate)
' oldWeekDay now contains 4 because thisDate represents a Wednesday.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Day, fonction (Visual Basic)
Month, fonction (Visual Basic)
Now, propriété
WeekdayName, fonction (Visual Basic)
Year, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
DateTime

WeekdayName, fonction (Visual Basic)

Retourne une valeur de type **String** contenant le nom du jour de semaine spécifié.

```
Public Function WeekdayName( _
   ByVal Weekday As Integer, _
   Optional ByVal Abbreviate As Boolean = False, _
   Optional ByVal FirstDayOfWeekValue As FirstDayOfWeek = FirstDayOfWeek.System _
) As String
```

Paramètres

Weekday

Requis. **Integer**. Désignation numérique du jour de la semaine, de 1 à 7 ; 1 représente le premier jour de la semaine et 7 le dernier. Les identités des premier et dernier jours dépendent du paramètre de *FirstDayOfWeekValue*.

Abbreviate

Valeur **Boolean** facultative qui indique si le nom du jour ouvrable doit être abrégé. Si cette valeur est omise, la valeur par défaut est **False**, ce qui signifie que le nom du jour ouvrable n'est pas abrégé.

FirstDayOfWeekValue

Facultatif. Valeur choisie dans l'énumération **FirstDayOfWeek** qui spécifie le premier jour de la semaine. S'il n'est pas spécifié, **FirstDayOfWeek.System** est utilisé.

Paramètres

L'argument *FirstDayOfWeekValue* peut prendre l'une des valeurs suivantes :

Valeur d'énumération	Valeur	Description
FirstDayOfWeek.System	0	Premier jour de la semaine spécifiée dans les paramètres système (valeur par défaut)
FirstDayOfWeek.Sunday	1	Dimanche
FirstDayOfWeek.Monday	2	Lundi (conformément à la norme ISO 8601, section 3.17)
FirstDayOfWeek.Tuesday	3	Mardi
FirstDayOfWeek.Wednesday	4	Mercredi
FirstDayOfWeek.Thursday	5	Jeudi
FirstDayOfWeek.Friday	6	Vendredi
FirstDayOfWeek.Saturday	7	Samedi

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Weekday* est inférieur à 1 ou supérieur à 7 ou *FirstDayOfWeekValue* est inférieur à 0 ou supérieur à 7.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La chaîne retournée par **WeekdayName** dépend non seulement des arguments d'entrée, mais également des paramètres **Options régionales** spécifiés dans le **Panneau de configuration** Windows.

Remarque

WeekdayName utilise le paramètre de calendrier actuel de la propriété CurrentCulture de la classe CultureInfo dans l'espace de noms System.Globalization. Les valeurs par défaut de **CurrentCulture** dépendent des paramètres du **Panneau de configuration**.

Exemple

L'exemple suivant utilise la fonction **Weekday** pour obtenir le jour de la semaine à partir d'une date spécifiée, puis la fonction **WeekDayName** pour obtenir le nom du jour ouvrable à partir de son numéro.

VB

```
Dim oldDate As Date
Dim oldWeekDayName As String
oldDate = #2/12/1969#
oldWeekDayName = WeekdayName(Weekday(oldDate))
' oldWeekDayName now contains "Wednesday".
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Day, fonction (Visual Basic)
Month, fonction (Visual Basic)
Now, propriété
Weekday, fonction (Visual Basic)
MonthName, fonction (Visual Basic)
Year, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
DateTime

Fonctions Write, WriteLine

Écrit des données dans un fichier séquentiel. Les données écrites à l'aide de la fonction **Write** sont généralement lues à partir d'un fichier avec la fonction **Input**.

```
Public Sub Write( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output As Object _
)
' -or-
Public Sub WriteLine( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output() As Object _
)
```

Paramètres

FileNumber

Requis. Expression de type **Integer** contenant tout numéro de fichier valide.

Output

Facultatif. Une ou plusieurs expressions séparées par des virgules à écrire dans un fichier.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
IOException	52	*FileNumber* n'existe pas.
IOException	54	Le mode de fichier est non valide.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Les fonctions **Write** et **WriteLine** sont fournies pour une compatibilité descendante et peut avoir un impact sur les performances. Pour les applications non héritées (legacy), l'objet **My.Computer.FileSystem** offre de meilleures performances. Pour plus d'informations, consultez Accès au fichier avec Visual Basic.

Si l'argument *Output* est omis, une ligne vide est imprimée dans le fichier. Les expressions multiples peuvent être séparées par une virgule.

Contrairement à la fonction **Print**, la fonction **Write** insère des virgules entre les éléments et des guillemets de part et d'autre des chaînes au moment de leur écriture dans le fichier. Vous n'avez donc pas à placer de délimiteurs explicites dans la liste. Lorsque la fonction **Write** est utilisée pour écrire des données dans un fichier, seuls les formats de données numériques, **Boolean**, de date, null et de **Error** sont prises en charge. Les hypothèses universelles suivantes sont suivies afin que les données puissent toujours être lues et correctement interprétées à l'aide de **Input**, indépendamment des paramètres régionaux :

- Les données numériques sont toujours écrites avec un point comme séparateur décimal.
- Pour les données de type **Boolean**, les valeurs `#TRUE#` ou `#FALSE#` sont imprimées. Les mots clés **True** et **False** ne sont jamais traduits, quels que soient les paramètres régionaux.
- Les données relatives aux dates sont écrites dans le fichier conformément au format de date universel. Si l'un des composants date ou heure est omis ou égal à zéro, seul le composant fourni est écrit dans le fichier.
- Si les données de *Output* sont vides, rien n'est écrit dans le fichier. Toutefois, dans le cas de données de type null, la valeur `#NULL#` est écrite.
- Pour les données de type **Error**, le résultat affiché est `#ERROR errorcode#`. Le mot clé **Error** n'est jamais traduit, quels que

soient les paramètres régionaux.

La fonction **WriteLine** insère un caractère de passage à la ligne, c'est-à-dire un retour chariot/saut de ligne (`Chr(13) + Chr(10)`), après l'écriture dans le fichier du dernier caractère contenu dans l'argument *Output*.

Vous pouvez insérer des guillemets dans une chaîne en utilisant des guillemets doubles, ou "". Par exemple :

VB

```
Dim x As String = "Double quotation marks aren't ""difficult"" to handle."
```

retourne une chaîne avec la valeur de `Double quotation marks aren't "difficult" to handle`.

L'écriture sur fichier à l'aide de la fonction **Write** ou **WriteLine** requiert l'accès **Append** à partir de l'énumération **FileIOPermissionAccess**. Pour plus d'informations, consultez FileIOPermissionAccess, énumération.

Exemple

Cet exemple utilise la fonction **Write** pour écrire des données brutes dans un fichier séquentiel.

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output) ' Open file for output.
Write(1, "This is a test.")  ' Print text to file.
WriteLine(1)  ' Print blank line to file.
WriteLine(1, "Zone 1", TAB(), "Zone 2")   ' Print in two print zones.
WriteLine(1, "Hello", " ", "World")     ' Separate strings with space.
WriteLine(1, SPC(5), "5 leading spaces ")    ' Print five leading spaces.
WriteLine(1, TAB(10), "Hello")   ' Print word at column 10.

' Assign Boolean, Date, and Error values.
Dim aBool As Boolean
Dim aDate As DateTime
aBool = False
aDate = DateTime.Parse("February 12, 1969")

' Dates and Booleans are translated using locale settings of
' your system.
WriteLine(1, aBool, " is a Boolean value")
WriteLine(1, aDate, " is a date")
FileClose(1)   ' Close file.
```

Notes du développeur sur Smart Device

Cette fonction n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : FileSystem

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : insérer du texte dans des fichiers dans Visual Basic

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Référence

Input, fonction

FileOpen, fonction

Fonctions Print, PrintLine

Autres ressources

Accès au fichier avec Visual Basic

Year, fonction (Visual Basic)

Retourne une valeur de type **Integer** comprise entre 1 et 9999 représentant l'année.

```
Public Function Year(ByVal DateValue As DateTime) As Integer
```

Paramètres
DateValue

Requis. Valeur **Date** de laquelle vous souhaitez extraire l'année.

Notes

Vous pouvez également obtenir l'année en appelant **DatePart** et en spécifiant **DateInterval.Year** pour l'argument *Interval*.

Exemple

L'exemple suivant utilise la fonction **Year** pour obtenir l'année à partir d'une date spécifiée. Dans l'environnement de développement, le littéral de date est affiché sous le format de date courte en utilisant les paramètres régionaux de votre code.

VB

```
Dim thisDate As Date
Dim thisYear As Integer
thisDate = #2/12/1969#
thisYear = Year(thisDate)
' thisYear now contains 1969.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Day, fonction (Visual Basic)
Month, fonction (Visual Basic)
Now, propriété
Weekday, fonction (Visual Basic)
DatePart, fonction (Visual Basic)
DateTime
ArgumentOutOfRangeException

Mots clés (Visual Basic)

Les rubriques de cette section contiennent des tableaux des mots clés du langage Visual Basic, réservés et non réservés.

Dans cette section

Mots clés A-E

Mots clés F-O

Mots clés P-Z

Rubriques connexes

Référence Visual Basic

Visual Basic

Mots clés A-E

Le tableau suivant énumère les mots clés du langage Visual Basic.

Alias	AddHandler	Ansi	As
Assembly	Auto	Binary	ByRef
ByVal	Case	Catch	Class
Custom	Default	DirectCast	Each
Else	ElseIf	End	Error

Notes du développeur sur Smart Device

Les éléments suivants ne sont pas pris en charge dans les applications Smart Device : **Ansi**, **Auto**, et **End**.

Voir aussi

Référence

Mots clés F-O

Mots clés P-Z

Autres ressources

Référence Visual Basic

Alias

Indique qu'une procédure externe possède un autre nom dans sa DLL.

Notes

Le mot clé **Alias** peut être utilisé dans le contexte suivant :

Declare, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

AddHandler

AddHandler associe un événement à un gestionnaire d'événements au moment de l'exécution, ou déclare quel code exécuter lors de l'ajout d'un gestionnaire d'événements.

Notes

Le mot clé **AddHandler** est utilisé dans les contextes suivants :

Event, instruction

Le mot clé **AddHandler** déclare un accesseur **AddHandler** personnalisé.

AddHandler, instruction

Le mot clé **AddHandler** associe un événement à un gestionnaire d'événements.

Voir aussi

Tâches

Comment : déclarer des événements qui évitent les blocages

Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Ansi

Spécifie que Visual Basic doit marshaler toutes les chaînes en valeurs ANSI (American Standards Institute) indépendamment du nom de la procédure externe qui est déclarée.

Lorsque vous appelez une procédure définie à l'extérieur de votre projet, le compilateur Visual Basic n'a pas accès aux informations requises pour appeler la procédure correctement. Ces informations incluent l'emplacement de la procédure, son identification, sa séquence d'appel et son type de retour ainsi que le jeu de caractères de chaîne qu'elle utilise. L'instruction Declare crée une référence à une procédure externe et fournit ces informations nécessaires.

L'élément *charsetmodifier* de l'instruction **Declare** fournit les informations de jeu de caractères pour marshaler des chaînes lors d'un appel à la procédure externe. Elle affecte également la façon dont Visual Basic recherche le nom de la procédure externe dans le fichier externe. Le modificateur **Ansi** spécifie que Visual Basic doit marshaler toutes les chaînes en valeurs ANSI et doit rechercher la procédure sans modifier son nom lors de la recherche.

Si aucun modificateur de jeu de caractères n'est spécifié, **Ansi** s'applique par défaut.

Notes

Le modificateur **Ansi** peut être utilisé dans le contexte suivant :

Declare, instruction

Notes du développeur sur Smart Device

Ce mot clé n'est pas pris en charge.

Voir aussi

Référence

Auto
Unicode (Visual Basic)
Mots clés du langage Visual Basic

As (Visual Basic)

Introduit une clause **As** qui identifie un type de données dans une instruction de déclaration ou une liste de contraintes sur un paramètre de type générique.

Notes

Le mot clé **As** peut être utilisé dans les contextes suivants :

Class, instruction

Const, instruction

Declare, instruction

Delegate, instruction

Dim, instruction

Enum, instruction

Event, instruction

For...Next, instructions

For Each...Next, instructions

Function, instruction

Interface, instruction

Operator, instruction

Property, instruction

Structure, instruction

Sub, instruction

Try...Catch...Finally, instructions

Voir aussi

Tâches

Comment : créer une variable

Référence

Liste de types

Mots clés du langage Visual Basic

Concepts

Types de données en Visual Basic

Déclaration de variable dans Visual Basic

Types génériques en Visual Basic

Assembly

Spécifie qu'un attribut s'applique au début d'un fichier source à l'intégralité de l'assembly.

Notes

Beaucoup d'attributs appartiennent à un élément de programmation, tel qu'une classe ou une propriété. Vous appliquez ce type d'attribut en liant le bloc d'attributs, entre les signes "inférieur à" et "supérieur à" (**< >**), directement à l'instruction de déclaration.

Si un attribut n'appartient pas uniquement à l'élément suivant, mais aussi à l'intégralité de l'assembly, placez le bloc d'attributs au début du fichier source et identifiez l'attribut avec le mot clé **Assembly**. S'il s'applique au module d'assembly actuel, utilisez le mot clé Module.

Vous pouvez également appliquer un attribut à un assembly dans le fichier AssemblyInfo.vb, auquel cas vous n'êtes pas obligé d'utiliser un bloc d'attributs dans votre fichier de code source principal.

Voir aussi

Référence

Module (Visual Basic)

Concepts

Application des attributs

Auto

Spécifie que Visual Basic doit marshaler les chaînes d'après les règles de .NET Framework selon le nom externe de la procédure externe déclarée.

Lorsque vous appelez une procédure définie à l'extérieur de votre projet, le compilateur Visual Basic n'a pas accès aux informations requises pour appeler la procédure correctement. Ces informations incluent l'emplacement de la procédure, son identification, sa séquence d'appel et son type de retour ainsi que le jeu de caractères de chaîne qu'elle utilise. L'instruction Declare crée une référence à une procédure externe et fournit ces informations nécessaires.

La partie *charsetmodifier* de l'instruction **Declare** fournit les informations de jeu de caractères pour marshaler des chaînes lors d'un appel à la procédure externe. Elle affecte également la façon dont Visual Basic recherche le nom de la procédure externe dans le fichier externe. Le modificateur **Auto** spécifie que Visual Basic doit marshaler les chaînes d'après les règles de .NET Framework et qu'il doit déterminer le jeu de caractères de base de la plate-forme d'exécution et éventuellement modifier le nom de procédure externe si la recherche initiale échoue. Pour plus d'informations, consultez « Jeux de caractères » dans la rubrique relative à l'instruction Declare.

Si aucun modificateur de jeu de caractères n'est spécifié, **Ansi** s'applique par défaut.

Notes

Le modificateur **Auto** peut être utilisé dans le contexte suivant :

Declare, instruction

Notes du développeur sur Smart Device

Ce mot clé n'est pas pris en charge.

Voir aussi

Référence

Ansi
Unicode (Visual Basic)
Mots clés du langage Visual Basic

Binary

Définit la méthode de comparaison de chaînes à un ordre de tri binaire strict.

Notes

L'instruction **Option Compare** doit apparaître dans un fichier source avant toute autre instruction. Cela affecte la comparaison de chaînes pour tout le code généré par ce fichier source.

Quand utiliser Binary

Vous devrez peut-être comparer des éléments de chaîne basés sur leur valeur binaire true, en particulier si les chaînes peuvent contenir des caractères qui ne seront pas interprétés comme du texte. Dans ce cas, vous ne souhaitez pas influencer des comparaisons avec les équivalences alphabétiques telles que le non-respect de la casse. Pour spécifier une comparaison binaire pure, attribuez la valeur **Binary** à **Option Compare**.

Le mot clé **Binary** peut être utilisé dans le contexte suivant :

Option Compare, instruction

Voir aussi
Référence
Text
Mots clés du langage Visual Basic

ByRef

Spécifie qu'un argument est passé de telle manière que la procédure appelée ne peut pas changer la valeur d'une variable sous-jacente à l'argument dans le code appelant.

Notes

Le modificateur **ByRef** peut être utilisé dans les contextes suivants :

Declare, instruction

Function, instruction

Sub, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Concepts

Passage d'arguments par valeur et par référence

ByVal

Spécifie qu'un argument est passé de telle manière que la procédure ou la propriété appelée ne peut pas changer la valeur d'une variable sous-jacente à l'argument dans le code appelant.

Notes

Le modificateur **ByVal** peut être utilisé dans les contextes suivants :

Declare, instruction

Function, instruction (Visual Basic)

Operator, instruction

Property, instruction

Sub, instruction (Visual Basic)

Voir aussi

Référence

Mots clés du langage Visual Basic

Concepts

Passage d'arguments par valeur et par référence

Case (Visual Basic)

Introduit une valeur ou un ensemble de valeurs par rapport auxquelles la valeur d'une expression doit être testée.

Notes

Le mot clé **Case** peut être utilisé dans le contexte suivant :

Instruction Select...Case

Voir aussi

Référence

Mots clés du langage Visual Basic

Catch (Visual Basic)

Présente un bloc d'instruction à exécuter si une exception particulière se produit à l'intérieur d'un bloc **Try**.

Notes

Chaque structure **Try** doit inclure un bloc **Finally** ou au moins un bloc **Catch**.

Quand utiliser Catch

Si vous vous attendez à ce qu'une exception particulière se produise pendant une section particulière de code, mettez le code dans un bloc **Try** et utilisez un bloc **Catch** pour conserver le contrôle et gérer l'exception si elle se produit. Vous pouvez utiliser autant de blocs **Catch** différents que nécessaire dans une structure **Try**.

Le mot clé **Catch** peut être utilisé dans le contexte suivant :

Try...Catch...Finally, instruction (Visual Basic)

Voir aussi

Référence

Finally (Visual Basic)

Mots clés du langage Visual Basic

Class (Visual Basic)

Contraint un paramètre de type générique à demander que tous les arguments de type qui lui sont passés soient un type référence, ou introduit un Class, instruction (Visual Basic).

Notes

Lorsque vous déclarez un paramètre de type pour un type générique, vous pouvez imposer une *contrainte*, un jeu d'un ou plusieurs conditions requises qui limitent l'argument de type qui peut être passé à ce paramètre de type. Pour plus d'informations, consultez « Constraints » dans Types génériques en Visual Basic.

L'une des conditions requises possibles dans une contrainte est que l'argument de type doit être un type référence, par exemple un **String**, un tableau ou un délégué, ou un objet créé à partir d'une classe. Cette spécification s'effectue en incluant le mot clé **Class** dans la contrainte.

Une autre configuration requise possible est que l'argument de type doit être un type valeur, par exemple, une structure, une énumération ou un type de données élémentaire. Vous devez spécifier une condition requise de type valeur en incluant le mot clé Structure (Visual Basic) dans la contrainte.

Vous ne devez spécifier ni **Class** ni **Structure** dans une contrainte. Vous ne pouvez pas les spécifier à la fois dans la même contrainte.

La contrainte **Class** est différente de Class, instruction (Visual Basic).

Voir aussi

Référence

Structure (Visual Basic)

Class, instruction (Visual Basic)

Concepts

Types génériques en Visual Basic

Types valeur et types référence

Custom

Spécifie qu'un événement a du code supplémentaire spécialisé pour ajouter des gestionnaires, en supprimer et déclencher des événements.

Notes

Le mot clé **Custom** peut être utilisé dans le contexte suivant :

Event, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Default (Visual Basic)

Identifie une propriété comme propriété par défaut de sa classe, structure ou interface.

Notes

Une classe, structure ou interface peut désigner au plus l'une de ses propriétés comme *propriété par défaut*, à condition que la propriété prenne au moins un paramètre. Si le code fait une référence à une classe ou structure sans spécifier un membre, Visual Basic résout cette référence à la propriété par défaut.

Les propriétés par défaut peuvent entraîner une légère réduction des caractères de code source, mais elles peuvent rendre votre code plus difficile à lire. Si le code appelant ne connaît pas votre classe ou votre structure, lorsqu'il fait référence au nom de la classe ou de la structure, il ne peut pas déterminer avec certitude si cette référence accède à la classe ou à la structure elle-même ou à une propriété par défaut. Cela peut engendrer des erreurs du compilateur ou des erreurs d'exécution logiques subtiles.

Vous pouvez légèrement réduire la probabilité que se produisent des erreurs de propriété par défaut en utilisant toujours l'instruction Option Strict pour affecter la valeur **On** au contrôle de type de compilateur.

Si vous envisagez d'utiliser une classe ou une structure prédéfinie dans votre code, vous devez déterminer si elle a une propriété par défaut, et, le cas échéant, indiquer son nom.

À cause de ces inconvénients, vous devez envisager de ne pas définir de propriétés par défaut. Afin de garantir la lisibilité du code, vous devez également envisager de toujours faire référence explicitement à toutes les propriétés, y compris aux propriétés par défaut.

Le modificateur **Default** peut être utilisé dans le contexte suivant :

Property, instruction

Voir aussi

Tâches

Comment : déclarer et appeler une propriété par défaut en Visual Basic

Référence

Mots clés du langage Visual Basic

Concepts

Propriétés par défaut

DirectCast

Introduit une opération de conversion de type basée sur l'héritage ou l'implémentation.

Notes

DirectCast n'utilise pas les routines d'assistance d'exécution Visual Basic pour la conversion, par conséquent, il peut en quelque sorte donner de meilleures performances que **CType** lors de la conversion vers et à partir du type de données **Object**.

Utilisez le mot clé **DirectCast** de la même façon que si vous utilisiez les mots clés CType et TryCast. Fournissez une expression comme premier argument et un type pour la convertir comme deuxième argument. **DirectCast** requiert une relation d'héritage ou d'implémentation entre les types de données des deux arguments. Cela signifie qu'un type doit hériter de l'autre ou l'implémenter.

Erreurs et échecs

DirectCast génère une erreur du compilateur s'il détecte qu'il n'y a aucune relation d'héritage ou d'implémentation. Cependant, l'absence d'erreur du compilateur ne garantit pas une conversion réussie. Si la conversion souhaitée est restrictive, elle peut échouer au moment de l'exécution. Le cas échéant, l'exécution lève une erreur InvalidCastException.

Mots clés de conversion

Une comparaison des mots clés de conversion de type se présente comme suit.

Mot clé	Types de données	Relation d'argument	Échec lors de l'exécution
CType, fonction	Tout type de données	Une conversion étendue ou restrictive doit être définie entre les deux types de données	Lève **InvalidCastException**
DirectCast	Tout type de données	Un type doit hériter de l'autre type ou l'implémenter	Lève **InvalidCastException**
TryCast	Types référence uniquement	Un type doit hériter de l'autre type ou l'implémenter	Retourne Nothing (Visual Basic)

Exemple

L'exemple suivant illustre deux utilisations de **DirectCast**, une qui échoue au moment de l'exécution et une qui réussit.

VB

```
Dim q As Object = 2.37
Dim i As Integer = CType(q, Integer)
' The following conversion fails at run time
Dim j As Integer = DirectCast(q, Integer)
Dim f As New System.Windows.Forms.Form
Dim c As System.Windows.Forms.Control
' The following conversion succeeds.
c = DirectCast(f, System.Windows.Forms.Control)
```

Dans l'exemple précédent, le type d'exécution de `q` est **Double**. **CType** réussit parce que **Double** peut être converti en **Integer**. Toutefois, la première utilisation de **DirectCast** échoue au moment de l'exécution parce que le type d'exécution de **Double** n'a aucune relation d'héritage avec **Integer**, bien qu'une conversion existe. La deuxième utilisation de **DirectCast** réussit parce qu'elle convertit depuis le type Form vers le type Control duquel **Form** hérite.

Voir aussi

Concepts

Conversions étendues et restrictives

Conversions implicites et explicites

Each

Spécifie la variable de boucle à utiliser dans une boucle **For Each**.

Notes

Le mot clé **Each** est utilisé dans le contexte suivant :

For Each...Next, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Else (Visual Basic)

Présente un groupe d'instructions à exécuter ou à compiler si aucun autre groupe conditionnel d'instructions n'a été exécuté ou compilé.

Notes

Le mot clé **Else** peut être utilisé dans les contextes suivants :

If...Then...Else, instruction

Select...Case, instruction

#If...Then...#Else, directive

Voir aussi

Référence

Mots clés du langage Visual Basic

ElseIf (Visual Basic)

Introduit une condition à tester si le test conditionnel précédent a échoué.

Notes

Le mot clé **ElseIf** peut être utilisé dans les contextes suivants :

Instruction If...Then...Else

Directive #If...Then...#Else

Voir aussi

Référence

Mots clés du langage Visual Basic

End (Visual Basic)

Lorsqu'il est suivi par un mot clé supplémentaire, il termine la définition du bloc d'instruction introduit par ce mot clé.

```
End AddHandler
End Class
End Enum
End Event
End Function
End Get
End If
End Interface
End Module
End Namespace
End Operator
End Property
End RaiseEvent
End RemoveHandler
End Select
End Set
End Structure
End Sub
End SyncLock
End Try
End While
End With
```

Éléments

End

Requis. Termine la définition de l'élément de programmation.

AddHandler

Requis pour terminer un accesseur **AddHandler** commencé par une instruction **AddHandler** correspondante dans une instruction Event personnalisée.

Class

Requis pour terminer une définition de classe commencée par une instruction Class (Visual Basic) correspondante.

Enum

Requis pour terminer une définition d'énumération commencée par une instruction Enum (Visual Basic) correspondante.

Event

Requis pour terminer une définition d'événement Custom commencée par une instruction Event correspondante.

Function

Requis pour terminer une définition de procédure **Function** commencée par une instruction Function (Visual Basic) correspondante. Si l'exécution rencontre une instruction **End Function**, le contrôle est retourné au code appelant.

Get

Requis pour terminer une définition de procédure **Property** commencée par une instruction Get correspondante. Si l'exécution rencontre une instruction **End Get**, le contrôle est retourné à l'instruction demandant la valeur de la propriété.

If

Requis pour terminer une définition de bloc **If**...**Then**... **Else** commencée par une instruction **If** correspondante. Consultez If...Then...Else, instruction (Visual Basic).

Interface

Requis pour terminer une définition d'interface commencée par une instruction Interface (Visual Basic) correspondante.

Module

Requis pour terminer une définition de module commencée par une instruction Module correspondante.

Namespace

Requis pour terminer une définition d'espace de noms commencée par une instruction Namespace correspondante.

Operator

Requis pour terminer une définition d'opérateur commencée par une instruction Operator correspondante.

Property

Requis pour terminer une définition de propriété commencée par une instruction Property correspondante.

RaiseEvent

Requis pour terminer un accesseur **RaiseEvent** commencé par une instruction **RaiseEvent** correspondante dans une instruction Event personnalisée.

RemoveHandler

Requis pour terminer un accesseur **RemoveHandler** commencé par une instruction **RemoveHandler** correspondante dans une instruction Event personnalisée.

Select

Requis pour terminer une définition de bloc **Select**...**Case** commencée par une instruction **Select** correspondante. Consultez Select...Case, instruction (Visual Basic).

Set

Requis pour terminer une définition de procédure **Property** commencée par une instruction Set (Visual Basic) correspondante. Si l'exécution rencontre une instruction **End Set**, le contrôle est retourné à l'instruction définissant la valeur de la propriété.

Structure

Requis pour terminer une définition de structure commencée par une instruction Structure correspondante.

Sub

Requis pour terminer une définition de procédure **Sub** commencée par une instruction Sub (Visual Basic) correspondante. Si l'exécution rencontre une instruction **End Sub**, le contrôle est retourné au code appelant.

SyncLock

Requis pour terminer une définition de bloc **SyncLock** commencée par une instruction **SyncLock** correspondante. Consultez SyncLock, instruction.

Try

Requis pour terminer une définition de bloc **Try**...**Catch**...**Finally** commencée par une instruction **Try** correspondante. Consultez Try...Catch...Finally, instruction (Visual Basic).

While

Requis pour terminer une définition de boucle **While** commencée par une instruction **While** correspondante. Consultez While...End While, instruction (Visual Basic).

With

Requis pour terminer une définition de bloc **With** commencée par une instruction **With** correspondante. Consultez With...End With, instruction (Visual Basic).

Notes

L'instruction End, sans mot clé supplémentaire, termine l'exécution immédiatement.

Lorsque précédé par un signe dièse (**#**), le mot clé **End** termine un bloc prétraitant introduit par la directive correspondante.

#End

Requis. Termine la définition du bloc de prétraitement.

#ExternalSource

Requis pour terminer un bloc source externe commencé par une directive #ExternalSource correspondante.

#If

Requis pour terminer un bloc de compilation conditionnel commencé par une directive **#If** correspondante. Consultez #If...Then...#Else, directives.

#Region

Requis pour terminer un bloc source region commencé par une directive #Region correspondante.

Notes du développeur sur Smart Device

L'instruction **End**, sans mot clé supplémentaire, n'est pas prise en charge.

Voir aussi

Référence

End, instruction

Error (Visual Basic)

Simule ou répond à une erreur d'exécution.

Notes

Le mot clé **Error** peut être utilisé dans les contextes suivants :

Error, instruction

On Error, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

Mots clés F-O

Le tableau suivant énumère les mots clés du langage Visual Basic.

False	Finally	For	Friend
Global	Handles	Implements	In
Is	Lib	Loop	Me
Module	MustInherit	MustOverride	MyBase
MyClass	Narrowing	New	Next
Nothing	NotInheritable	NotOverridable	Of
Off	On	Option	Optional
Overloads	Overridable	Overrides	

Voir aussi

Référence

Mots clés A-E

Mots clés P-Z

Autres ressources

Référence Visual Basic

False (Visual Basic)

Représente une valeur **Boolean** qui échoue à un test conditionnel.

Notes

Les valeurs Boolean, type de données (Visual Basic) ne sont pas stockées en tant que nombres et les valeurs stockées ne sont pas destinées à être équivalentes aux nombres. Vous ne devez jamais écrire un code qui repose sur les valeurs numériques équivalentes à **True** et **False**. Autant que possible, limitez l'utilisation des variables **Boolean** aux valeurs logiques pour lesquelles elles sont conçues. S'il s'avère nécessaire de mélanger des valeurs **Boolean** et des valeurs numériques, assurez-vous que vous comprenez la méthode de conversion choisie. Pour plus d'informations, consultez « Le type Booléen ne se convertit pas correctement en type numérique » dans Dépannage des types de données.

Voir aussi

Tâches

Dépannage des types de données

Référence

Boolean, type de données (Visual Basic)

True (Visual Basic)

Finally (Visual Basic)

Introduit un bloc d'instructions à exécuter avant de quitter une structure **Try**.

Notes

Chaque structure **Try** doit inclure un bloc **Finally** ou au moins un bloc **Catch**.

Remarque
Le contrôle ne passe pas d'un bloc **Try** ou **Catch** au bloc **Finally** correspondant dans les cas suivants :

- L'instruction End existe dans le bloc **Try** ou **Catch**.
- Un StackOverflowException est levé dans le bloc **Try** ou **Catch**.

Quand utiliser Finally

Si une ou plusieurs instructions doivent être exécutées avant que vous ne sortiez de la structure **Try**, utilisez un bloc **Finally**. Le contrôle passe au bloc **Finally** juste avant de sortir de la structure **Try**. Cela est vrai même si une exception se produit où que ce soit dans la structure **Try**.

Le mot clé **Finally** peut être utilisé dans le contexte suivant :

Try...Catch...Finally, instruction (Visual Basic)

Voir aussi
Référence
Catch (Visual Basic)
Mots clés du langage Visual Basic

For (Visual Basic)

Introduit une boucle parcourue par diverses valeurs d'une variable de boucle.

Notes

Le mot clé **For** est utilisé dans les contextes suivants :

For...Next, instructions

For Each...Next, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Friend (Visual Basic)

Spécifie qu'un ou plusieurs éléments de programmation déclarés sont accessibles uniquement à partir de l'assembly qui contient leur déclaration.

Notes
Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Friend** seulement au niveau du module, de l'interface ou de l'espace de noms. Cela signifie que le contexte de déclaration pour un élément **Friend** doit être un fichier source, un espace de noms, un module, une classe ou une structure, et ne peut pas être une procédure.
- **Modificateurs combinés.** Vous pouvez utiliser le modificateur **Friend** conjointement avec le modificateur Protected (Visual Basic) dans la même déclaration. Dans la mesure où cette combinaison confère à la fois l'accès ami et l'accès protégé aux éléments déclarés, ces derniers sont accessibles à partir du même assembly, de leur propre classe et de classes dérivées quelconques. Vous pouvez spécifier **Protected Friend** uniquement sur les membres d'une classe.

Comportement

- **Niveau d'accès.** Tout le code d'un contexte de déclaration peut accéder à ses éléments. Le code d'autres classes, structures et modules compilés dans le même assembly peut accéder à tous les éléments **Friend** de cet assembly.

  L'accès ami n'est pas un sur-ensemble ou sous-ensemble d'accès protégé.
- **Modificateurs d'accès.** Les mots clés qui spécifient le niveau d'accès portent le nom de *access modifiers*. Pour obtenir une comparaison entre les modificateurs d'accès, consultez Niveaux d'accès dans Visual Basic.

Quand utiliser Friend

Dans de nombreux cas, vous souhaitez que les éléments de programmation, tels que les classes et les structures, soient utilisés par l'assembly entier, pas uniquement par le composant qui les déclare. Toutefois, il se peut que vous souhaitiez les rendre inaccessibles par leur code à l'extérieur de l'assembly, par exemple si l'application est propriétaire. Si vous souhaitez limiter l'accès à un élément de cette manière, vous pouvez le déclarer avec **Friend**.

L'accès ami est souvent le niveau par défaut pour les éléments de programmation d'une application. Notez que le niveau d'accès d'une interface, d'un module, d'une classe, ou d'une structure prend par défaut la valeur **Friend** si vous ne le déclarez pas.

Le modificateur **Friend** peut être utilisé dans les contextes suivants :

Class, instruction

Const, instruction

Declare, instruction

Delegate, instruction

Dim, instruction

Enum, instruction

Event, instruction

Function, instruction

Interface, instruction

Module, instruction

Property, instruction

Structure, instruction

Sub, instruction

Voir aussi
Référence
Public (Visual Basic)

Protected (Visual Basic)
Private (Visual Basic)

Concepts

Niveaux d'accès dans Visual Basic
Procédures dans Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données
Présentation des classes

Global

Vous permet d'accéder à un élément de programmation .NET Framework lorsque vous l'avez bloqué avec votre structure d'espace de noms.

Notes

Si vous avez défini une hiérarchie imbriquée d'espaces de noms, le code à l'intérieur de cette hiérarchie peut ne pas pouvoir accéder à l'espace de noms System du .NET Framework. L'exemple suivant illustre une hiérarchie dans laquelle l'espace de noms `SpecialSpace.System` bloque l'accès à **System**.

```
Namespace SpecialSpace
    Namespace System
        Class abc
            Function getValue() As System.Int32
                Dim n As System.Int32
                Return n
            End Function
        End Class
    End Namespace
End Namespace
```

En conséquence, le compilateur Visual Basic ne peut pas résoudre correctement la référence à System.Int32, parce que `SpecialSpace.System` ne définit pas `Int32`. Vous pouvez utiliser le mot clé **Global** pour démarrer la chaîne de qualification au niveau le plus externe de la bibliothèque de classes .NET Framework. Cela vous permet de spécifier l'espace de noms **System** ou tout autre espace de noms dans la bibliothèque de classes. L'exemple suivant illustre ce comportement :

```
Namespace SpecialSpace
    Namespace System
        Class abc
            Function getValue() As Global.System.Int32
                Dim n As Global.System.Int32
                Return n
            End Function
        End Class
    End Namespace
End Namespace
```

Vous pouvez utiliser **Global** pour accéder à d'autres espaces de noms à la racine, tels que Microsoft.VisualBasic, et tout espace de noms associé à votre projet.

Le mot clé **Global** peut être utilisé dans les contextes suivants :

Class, instruction (Visual Basic)

Const, instruction (Visual Basic)

Declare, instruction

Delegate, instruction

Dim, instruction (Visual Basic)

Enum, instruction (Visual Basic)

Event, instruction

For...Next, instruction (Visual Basic)

For Each...Next, instruction (Visual Basic)

Function, instruction (Visual Basic)

Interface, instruction (Visual Basic)

Operator, instruction

Property, instruction

Structure, instruction

Sub, instruction (Visual Basic)

Try...Catch...Finally, instruction (Visual Basic)

Using, instruction (Visual Basic)

Voir aussi

Référence

Namespace, instruction

Référence de la bibliothèque de classes .NET Framework

System

Microsoft.VisualBasic

Handles

Déclare qu'une procédure gère un événement spécifié.

```
proceduredeclaration Handles eventlist
```

Éléments
proceduredeclaration

Déclaration de procédure **Sub** de la procédure qui gérera l'événement.

eventlist

Liste des événements de *proceduredeclaration* à gérer. Ces événements doivent être déclenchés par la classe de base de la classe en cours ou par un objet déclaré à l'aide du mot clé **WithEvents**.

Notes

Utilisez le mot clé **Handles** à la fin d'une déclaration de procédure pour que celle-ci gère les événements déclenchés par une variable objet déclarée à l'aide du mot clé **WithEvents**. Le mot clé **Handles** peut également être utilisé dans une classe dérivée pour gérer des événements à partir d'une classe de base.

La signature de la procédure doit correspondre aux signatures de chaque événement dans *eventlist*.

Le mot clé **Handles** et l'instruction **AddHandler** vous permettent de spécifier que des procédures particulières gèrent des événements particuliers, mais il existe des différences. Utilisez le mot clé **Handles** lors de la définition d'une procédure pour indiquer qu'elle gère un événement particulier. L'instruction **AddHandler** connecte des procédures aux événements au moment de l'exécution. Pour plus d'informations, consultez AddHandler, instruction.

Pour les événements personnalisés, l'application appelle l'accesseur **AddHandler** de l'événement lors de l'ajout de la procédure comme gestionnaire d'événements. Pour plus d'informations sur les événements personnalisés, consultez Event, instruction.

Exemple
VB

```
Public Class ContainerClass
    ' Module or class level declaration.
    WithEvents Obj As New Class1

    Public Class Class1
        ' Declare an event.
        Public Event Ev_Event()
        Sub CauseSomeEvent()
            ' Raise an event.
            RaiseEvent Ev_Event()
        End Sub
    End Class

    Sub EventHandler() Handles Obj.Ev_Event
        ' Handle the event.
        MsgBox("EventHandler caught event.")
    End Sub

    ' Call the TestEvents procedure from an instance of the ContainerClass
    ' class to test the Ev_Event event and the event handler.
    Public Sub TestEvents()
        Obj.CauseSomeEvent()
    End Sub
End Class
```

L'exemple suivant montre comment une classe dérivée peut utiliser l'instruction **Handles** pour gérer un événement à partir d'une classe de base.

VB

```
Public Class BaseClass
    ' Declare an event.
    Event Ev1()
End Class
Class DerivedClass
    Inherits BaseClass
    Sub TestEvents() Handles MyBase.Ev1
        ' Add code to handle this event.
    End Sub
End Class
```

Voir aussi

Référence

WithEvents

AddHandler, instruction

RemoveHandler, instruction

Event, instruction

RaiseEvent, instruction

Concepts

Événements et gestionnaires d'événements

Implements (Visual Basic)

Indique qu'un membre de classe ou de structure fournit l'implémentation d'un membre défini dans une interface.

Notes

Le mot clé **Implements** est différent de l'instruction Implements. Utilisez l'instruction **Implements** pour spécifier qu'une classe ou une structure implémente une ou plusieurs interfaces ; utilisez ensuite pour chaque membre le mot clé **Implements** pour spécifier quelle interface et quel membre il implémente.

Si une classe ou une structure implémente une interface, elle doit inclure l'instruction **Implements** directement après l'instruction Class (Visual Basic) ou Structure et doit implémenter tous les membres définis par l'interface.

Nouvelle implémentation

Dans une classe dérivée, vous pouvez implémenter de nouveau un membre d'interface que la classe de base a déjà implémenté. Cette opération est différente de la substitution du membre de la classe de base à plusieurs égards :

- Le membre de la classe de base ne doit pas être Overridable pour être de nouveau implémenté.
- Vous pouvez implémenter de nouveau le membre avec un nom différent.

Le mot clé **Implements** peut être utilisé dans les contextes suivants :

Event, instruction

Function, instruction (Visual Basic)

Property, instruction

Sub, instruction (Visual Basic)

Voir aussi

Référence

Implements, instruction
Interface, instruction (Visual Basic)
Class, instruction (Visual Basic)
Structure, instruction

Concepts

Implements, mot clé et instruction

In (Visual Basic)

Spécifie le groupe que la variable de boucle doit parcourir dans une boucle **For Each**.

Notes

Le mot clé **In** peut être utilisé dans le contexte suivant :

For Each...Next, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Is (Visual Basic)

Introduit une clause **Is** qui établit une comparaison.

Notes

Le mot clé **Is** est utilisé avec le mot clé **TypeOf** dans une expression **TypeOf**...**Is** pour tester si le type au moment de l'exécution d'une expression est compatible avec un type donné.

Le mot clé **Is** peut être utilisé dans les contextes suivants :

Is, opérateur

Select...Case, instruction

Voir aussi

Référence

TypeOf, opérateur (Visual Basic)

Mots clés du langage Visual Basic

Lib

Introduit une clause **Lib** qui identifie le fichier externe (DLL ou ressource de code) contenant une procédure externe.

Notes

Le mot clé **Lib** peut être utilisé dans le contexte suivant :

Declare, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Loop

Termine une boucle introduite par une instruction **Do**.

Notes

Le mot clé **Loop** peut être utilisé dans le contexte suivant :

Instruction Do...Loop

Voir aussi

Référence

Mots clés du langage Visual Basic

Me

Permet de faire référence à l'instance spécifique d'une classe ou d'une structure dans laquelle le code est en cours d'exécution.

Notes

Le mot clé **Me** se comporte soit comme une variable objet, soit comme une variable de structure faisant référence à l'instance en cours. L'utilisation de **Me** est particulièrement utile pour le passage des informations concernant l'instance d'une classe ou d'une structure en cours d'exécution, à une procédure se trouvant dans une autre classe, une autre structure ou un autre module. Par exemple, supposons que vous ayez la procédure suivante dans un module.

```
Sub ChangeFormColor(FormName As Form)
   Randomize()
   FormName.BackColor = Color.FromArgb(Rnd() * 256, Rnd() * 256, Rnd() * 256)
End Sub
```

Vous pouvez appeler cette procédure et passer l'instance active de la classe Form en tant qu'argument à l'aide de l'instruction suivante.

```
ChangeFormColor(Me)
```

Le mot clé **Me** peut être utilisé dans les contextes suivants :

Class, instruction

Structure, instruction

Voir aussi

Référence

MyBase

MyClass

Concepts

Assignation des variables objets

Module (Visual Basic)

Spécifie qu'un attribut au début d'un fichier source s'applique au module d'assembly actuel.

Notes

Beaucoup d'attributs appartiennent à un élément de programmation, tel qu'une classe ou une propriété. Vous appliquez ce type d'attribut en liant le bloc d'attributs, entre les signes "inférieur à" et "supérieur à" (**< >**), directement à l'instruction de déclaration.

Si un attribut n'appartient pas uniquement à l'élément suivant, mais aussi à l'assembly actuel, placez le bloc d'attributs au début du fichier source et identifiez l'attribut avec le mot clé **Module**. S'il s'applique à l'intégralité de l'assembly, utilisez le mot clé Assembly.

Le modificateur **Module** est différent de l'instruction Module.

Voir aussi

Référence

Assembly

Module, instruction

Concepts

Application des attributs

MustInherit

Spécifie qu'une classe peut être utilisée uniquement comme classe de base et que vous ne pouvez pas créer un objet directement à partir de cette classe.

Notes

Le but d'une *classe de base* (également connue sous le nom *classe abstraite*) est de définir une fonctionnalité commune à toutes les classes qui en sont dérivées. Cela évite aux classes dérivées de devoir redéfinir les éléments communs. Cette fonctionnalité commune n'est parfois pas suffisamment complète pour faire un objet utilisable, et chaque classe dérivée définit la fonctionnalité manquante. Dans ce cas, vous souhaitez que le code utilisateur crée uniquement des objets à partir des classes dérivées. Utilisez **MustInherit** sur la classe de base pour appliquer ceci.

Une autre utilisation d'une classe **MustInherit** consiste à restreindre une variable à un jeu de classes connexes. Vous pouvez définir une classe de base et en dériver toutes ces classes connexes. La classe de base n'a pas besoin de fournir une fonctionnalité commune à toutes les classes dérivées, mais peut servir de filtre pour assigner des valeurs aux variables. Si votre code utilisateur déclare une variable comme classe de base, Visual Basic vous permet d'assigner uniquement objet de l'une des classes dérivées à cette variable.

.NET Framework définit plusieurs classes **MustInherit** dont Array, Enum et ValueType. **ValueType** est un exemple de classe de base qui restreint une variable. Tous les types valeur dérivent de **ValueType**. Si vous déclarez une variable comme **ValueType**, vous ne pouvez assigner que des types valeur à cette variable.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **MustInherit** uniquement dans une instruction **Class**.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **MustInherit** avec **NotInheritable** dans la même déclaration.

Exemple

L'exemple suivant illustre à la fois l'héritage et la substitution forcés. La classe de base `shape` définit une variable `acrossLine`. Les classes `circle` et `square` dérivent de `shape`. Elles héritent de la définition de `acrossLine`, mais elles doivent définir la fonction `area`parce que ce calcul est différent pour chaque type de forme.

VB

```
Public MustInherit Class shape
    Public acrossLine As Double
    Public MustOverride Function area() As Double
End Class
Public Class circle : Inherits shape
    Public Overrides Function area() As Double
        Return Math.PI * acrossLine
    End Function
End Class
Public Class square : Inherits shape
    Public Overrides Function area() As Double
        Return acrossLine * acrossLine
    End Function
End Class
Public Class consumeShapes
    Public Sub makeShapes()
        Dim shape1, shape2 As shape
        shape1 = New circle
        shape2 = New square
    End Sub
End Class
```

Vous pouvez déclarer `shape1` et `shape2` comme type `shape`. Toutefois, vous ne pouvez pas créer un objet de `shape` parce que la fonctionnalité de la fonction `area` est manquante et il est marqué comme **MustInherit**.

Puisqu'elles sont déclarées comme `shape`, les variables `shape1` et `shape2` sont restreintes aux objets des classes dérivées `circle` et `square`. Visual Basic ne vous permet pas d'assigner un autre objet à ces variables, ce qui vous procure un niveau de sécurité de type élevé.

Utilisation

Le modificateur **MustInherit** peut être utilisé dans le contexte suivant :

Class, instruction

Voir aussi

Référence

Inherits, instruction

NotInheritable

Mots clés du langage Visual Basic

Autres ressources

Héritage en Visual Basic

MustOverride

Spécifie qu'une propriété ou une procédure n'est pas implémentée dans une classe de base et doit être substituée dans une classe dérivée avant son utilisation.

Notes

Vous pouvez utiliser **MustOverride** uniquement dans une propriété ou une instruction de déclaration de procédure. La propriété ou procédure qui spécifie **MustOverride** doit être un membre d'une classe, et la classe doit être marquée MustInherit.

Règles

- **Déclaration incomplète.** Lorsque vous spécifiez **MustOverride**, vous ne fournissez pas de lignes de code supplémentaires pour la propriété ou la procédure, ni l'instruction **End Function**, **End Property** ou **End Sub**.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **MustOverride** avec **NotOverridable**, **Overridable**, ou **Shared** dans la même déclaration.
- **Occultation et substitution.** L'occultation et la substitution redéfinissent toutes les deux un élément hérité, mais les deux approches sont significativement différentes. Pour plus d'informations, consultez Occultation dans Visual Basic.
- **Autres termes.** Un élément qui ne peut pas être utilisé sauf dans une substitution est parfois appelé élément *virtuel pur*.

Le modificateur **MustOverride** peut être utilisé dans les contextes suivants :

Function, instruction (Visual Basic)

Property, instruction

Sub, instruction (Visual Basic)

Voir aussi

Référence

NotOverridable
Overridable
Overrides
MustInherit
Mots clés du langage Visual Basic

Concepts

Occultation dans Visual Basic

MyBase

Permet de faire référence à la classe de base de l'instance de classe actuelle.

Notes

Le mot clé **MyBase** se comporte comme une variable objet faisant référence à la classe de base de l'instance actuelle d'une classe. **MyBase** est couramment utilisé pour accéder aux membres de la classe de base qui sont substitués ou occultés dans une classe dérivée. **MyBase.New** est, en particulier, utilisé pour appeler explicitement un constructeur de classe de base à partir d'un constructeur de classe dérivée.

L'utilisation de **MyBase** pour appeler les méthodes de base **MustOverride** n'est pas valide.

Pour obtenir des informations supplémentaires et un exemple, consultez
Comment : accéder à une variable masquée par une classe dérivée.

Voir aussi

Référence

Me
MyClass
New (Visual Basic)

Concepts

Éléments fondamentaux de l'héritage

MyClass

Permet de faire référence aux membres d'instance de la classe actuels sans qu'ils soient remplacés par toute substitution de classe dérivée.

Notes

Le mot clé **MyClass** se comporte comme une variable objet qui fait référence à l'instance actuelle d'une classe comme implémentée à l'origine. **MyClass** est semblable à Me, mais chaque appel de méthode et de propriété sur **MyClass** est traité comme si la méthode ou la propriété était NotOverridable. Par conséquent, la méthode ou la propriété n'est pas affectée par la substitution dans une classe dérivée. L'exemple suivant compare **Me** et **MyClass**.

```
Class baseClass
    Public Overridable Sub testMethod()
        MsgBox("Base class string")
    End Sub
    Public Sub useMe()
        ' The following call uses the calling class's version, even if
        ' that version is an override.
        Me.testMethod()
    End Sub
    Public Sub useMyClass()
        ' The following call uses this version and not any override.
        MyClass.testMethod()
    End Sub
End Class
Class derivedClass : Inherits baseClass
    Public Overrides Sub testMethod()
        MsgBox("Derived class string")
    End Sub
End Class
Class testClasses
    Sub startHere()
        Dim testObj As derivedClass = New derivedClass()
        ' The following call displays "Derived class string".
        testObj.useMe()
        ' The following call displays "Base class string".
        testObj.useMyClass()
    End Sub
End Class
```

Même si `derivedClass` substitue `testMethod`, le mot clé **MyClass** de `useMyClass` annule les effets de la substitution, et le compilateur résout l'appel vers la version de la classe de base de `testMethod`.

Vous ne pouvez pas utiliser **MyClass** dans une méthode Shared (Visual Basic), mais vous pouvez l'utiliser à l'intérieur d'une méthode d'instance pour accéder à un membre partagé d'une classe.

Voir aussi

Référence

Me

MyBase

Concepts

Éléments fondamentaux de l'héritage

Narrowing

Indique qu'un opérateur de conversion (**CType**) convertit une classe ou structure en un type qui ne pourra peut-être pas contenir certaines valeurs possibles de la classe d'origine ou structure.

Conversion avec le mot clé Narrowing

La procédure de conversion doit spécifier **Public Shared** en plus de **Narrowing**.

Les conversions restrictives ne réussissent pas toujours au moment de l'exécution et peuvent échouer ou entraîner la perte de données. Les exemples sont **Long** en **Integer**, **String** en **Date** et un type de base en un type dérivé. Cette dernière conversion est restrictive parce que le type de base ne pourra peut-être pas contenir tous les membres du type dérivé et donc n'est pas une instance du type dérivé.

Si **Option Strict** est **On**, le code utilisateur doit utiliser **CType** pour toutes les conversions restrictives.

Le mot clé **Narrowing** peut être utilisé dans le contexte suivant :

Operator, instruction

Voir aussi

Tâches

Comment : définir un opérateur

Référence

Operator, instruction
Widening
CType, fonction
Option Strict, instruction

Concepts

Conversions étendues et restrictives

New (Visual Basic)

Introduit une clause **New** pour créer une nouvelle instance d'objet ou spécifie une contrainte de constructeur sur un paramètre de type.

Notes

Dans une liste de paramètres de type, une contrainte **New** spécifie que le type fourni doit exposer un constructeur sans paramètre accessible. Pour plus d'informations sur les paramètres de type et les contraintes, consultez Liste de types.

Dans une instruction de déclaration ou d'assignation, une clause **New** doit spécifier une classe définie à partir de laquelle l'instance peut être créée. Cela signifie que la classe doit exposer un constructeur auquel le code appelant peut accéder.

Vous pouvez utiliser une clause **New** dans une instruction de déclaration ou une instruction d'assignation. Lorsque l'instruction est exécutée, elle appelle le constructeur de la classe spécifiée, en passant les arguments que vous avez fournis. C'est ce qu'illustre l'exemple suivant.

```
Dim someObj As Object
someObj = New someClass("String required by constructor")
Dim nextLabel As New Label()
```

Étant donné que les tableaux sont des classes, **New** peut créer une nouvelle instance de tableau, comme indiqué dans l'exemple suivant.

```
Dim intArray As Integer()
intArray = New Integer() {0, 1, 2, 3}
```

Le Common Language Runtime (CLR) lève une erreur OutOfMemoryException s'il n'y a pas assez de mémoire pour créer la nouvelle instance.

Le mot clé **New** peut être utilisé dans les contextes suivants :

Dim, instruction (Visual Basic)

Of

Voir aussi

Référence

Mots clés du langage Visual Basic
Liste de types
Utilisation de constructeurs et de destructeurs
OutOfMemoryException

Concepts

Types génériques en Visual Basic
Durée de vie d'un objet : création et destruction des objets

Next

Met fin à une boucle parcourue par diverses valeurs d'une variable de boucle ou spécifie que l'exécution doit se poursuivre avec l'instruction qui suit l'instruction ayant engendré l'erreur.

Notes

Le mot clé **Next** peut être utilisé dans les contextes suivants :

For...Next, instructions

For Each...Next, instruction

On Error, instruction

Resume, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Nothing (Visual Basic)

Représente la valeur par défaut d'un type de données.

Notes

L'assignation de **Nothing** à une variable la définit à sa valeur par défaut pour son type déclaré. Si ce type contient des membres de variable, ils ont tous leurs valeurs par défaut. L'exemple suivant illustre ce comportement :

```
Public Structure testStruct
    Public name As String
    Public number As Short
End Structure
Dim ts As testStruct, i As Integer, b As Boolean
ts = Nothing
' The preceding statement sets ts.name to "" and ts.number to 0.
i = Nothing
b = Nothing
' The preceding statements set i to 0 and b to False.
```

Si la variable est de type référence (une variable objet), **Nothing** signifie que la variable n'est pas associée à un objet. C'est ce qu'illustre l'exemple suivant.

```
Dim testObject As Object
testObject = Nothing
' The preceding statement sets testObject to not refer to any instance.
```

Lorsque vous assignez **Nothing** à une variable objet, cette dernière ne fait plus référence à une instance d'objet. Si la variable avait fait référence à une instance au préalable, l'assignation de **Nothing** à la variable ne met pas fin à l'instance. L'instance se termine, et les ressources mémoire et système qui lui sont associées sont libérées uniquement lorsque le garbage collector (GC) détecte l'absence de toute référence active restante.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Durée de vie d'un objet : création et destruction des objets

Durée de vie dans Visual Basic

NotInheritable

Spécifie qu'une classe ne peut pas être utilisée comme classe de base.

Notes

Autres termes. Une classe qui ne peut pas être héritée est parfois appelée classe *sealed*.

Le modificateur **NotInheritable** peut être utilisé dans le contexte suivant :

Class, instruction (Visual Basic)

Voir aussi

Référence

Inherits, instruction

MustInherit

Mots clés du langage Visual Basic

NotOverridable

Spécifie qu'une propriété ou une procédure ne peut pas être substituée dans une classe dérivée.

Notes

NotOverridable est le paramètre par défaut d'une propriété ou d'une procédure qui ne se substitue pas à une propriété ou une procédure de classe de base.

Un élément qui ne peut pas être substitué est parfois appelé élément *sealed*.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **NotOverridable** uniquement dans une propriété ou une instruction de déclaration de procédure. Vous pouvez spécifier **NotOverridable** uniquement sur une propriété ou une procédure qui substitue une autre propriété ou procédure, c'est-à-dire, uniquement en association avec **Overrides**.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **NotOverridable** avec **MustOverride**, **Overridable**, ou **Shared** dans la même déclaration.

Le modificateur **NotOverridable** peut être utilisé dans les contextes suivants :

Function, instruction (Visual Basic)

Property, instruction

Sub, instruction (Visual Basic)

Voir aussi

Référence

MustOverride
Overridable
Overrides
Mots clés du langage Visual Basic

Concepts

Occultation dans Visual Basic

Of

Introduit une clause **Of** qui identifie un *paramètre de type* sur une classe, une structure, une interface, un délégué ou une procédure *générique*. Pour plus d'informations sur les types génériques, consultez Types génériques en Visual Basic.

Utilisation du mot clé Of

L'exemple de code suivant utilise le mot clé **Of** pour définir le cadre d'une classe qui prend deux paramètres de type. Il *contraint* le paramètre `keyType` par l'interface IComparable, ce qui signifie que le code utilisateur doit fournir un argument de type qui implémente **IComparable**. Cela est nécessaire afin que la procédure `add` puisse appeler la méthode System.IComparable.CompareTo(System.Object). Pour plus d'informations sur les contraintes, consultez Liste de types.

```
Public Class Dictionary(Of entryType, keyType As IComparable)
    Public Sub add(ByVal e As entryType, ByVal k As keyType)
        Dim dk As keyType
        If k.CompareTo(dk) = 0 Then
        End If
    End Sub
    Public Function find(ByVal k As keyType) As entryType
    End Function
End Class
```

Si vous terminez la définition de la classe précédente, vous pouvez construire diverses classes `dictionary` à partir de celle-ci. Les types que vous fournissez à `entryType` et `keyType` déterminent le type d'entrée que la classe contient et le type de clé qu'elle associe à chaque entrée. À cause de la contrainte, vous devez fournir à `keyType` un type qui implémente **IComparable**.

L'exemple de code suivant crée un objet qui contient des entrées **String** et associe une clé **Integer** à chacune. **Integer** implémente **IComparable** et satisfait par conséquent la contrainte sur `keyType`.

```
Dim d As New dictionary(Of String, Integer)
```

Le mot clé **Of** peut être utilisé dans les contextes suivants :

Class, instruction

Delegate, instruction

Function, instruction

Interface, instruction

Structure, instruction

Sub, instruction

Voir aussi

Référence

Liste de types

IComparable

Concepts

Types génériques en Visual Basic

Off

Désactive une option du compilateur.

Notes

Le mot clé **Off** peut être utilisé dans les contextes suivants :

Option Explicit, instruction

Option Strict, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

On

Introduit une réponse à une erreur d'exécution ou active une option du compilateur.

Notes

Le mot clé **On** est utilisé dans les contextes suivants :

On Error, instruction

Option Explicit, instruction

Option Strict, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Option (Visual Basic)

Introduit une instruction qui spécifie une option du compilateur qui s'applique au fichier source entier.

Notes

Les options du compilateur peuvent vérifier si toutes les variables doivent être déclarées explicitement, si les conversions de type restrictives doivent être explicites ou si les chaînes doivent être comparées comme du texte ou comme des quantités binaires.

Le mot clé **Option** peut être utilisé dans les contextes suivants :

Option Compare, instruction

Option Explicit, instruction (Visual Basic)

Option Strict, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Optional (Visual Basic)

Spécifie qu'un argument de procédure peut être omis lorsque la procédure est appelée.

Notes

Le modificateur **Optional** peut être utilisé dans les contextes suivants :

Declare, instruction

Function, instruction

Property, instruction

Sub, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Overloads

Spécifie qu'une propriété ou une procédure redéclare une ou plusieurs propriétés ou procédures existantes avec le même nom.

Notes

La *surcharge* consiste à fournir plusieurs définitions pour une propriété ou un nom de procédure donné se trouvant dans la même portée. La redéclaration d'une propriété ou d'une procédure à l'aide d'une signature différente est parfois appelée *masquage par signature*.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Overloads** uniquement dans une propriété ou une instruction de déclaration de procédure.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **Overloads** avec Shadows dans la même déclaration de procédure.
- **Différences requises.** La *signature* dans cette déclaration doit être différente de la signature de chaque propriété ou procédure qu'elle surcharge. La signature comprend la propriété ou le nom de procédure avec ce qui suit :
    - le nombre de paramètres
    - l'ordre des paramètres
    - les types de données des paramètres
    - le nombre de paramètres de type (pour une procédure générique)
    - le type de retour (uniquement pour une procédure d'opérateur de conversion)

  Toutes les surcharges doivent avoir le même nom, mais chacune doit différer des autres à un ou plusieurs des égards ci-dessus. Cela permet au compilateur de distinguer la version à utiliser lorsque le code appelle la propriété ou la procédure.
- **Différences interdites.** La modification d'un ou de plusieurs des aspects suivants n'est pas valide pour la surcharge d'une propriété ou d'une procédure, parce qu'elles ne font pas partie de la signature :
    - une valeur est-elle retournée (pour une procédure)
    - le type de données de la valeur de retour (excepté pour un opérateur de conversion)
    - les noms des paramètres ou des paramètres de type
    - les contraintes sur les paramètres de type (pour une procédure générique)
    - mots clés de modificateur de paramètre (tels que **ByRef** et **Optional**)
    - mots clés de modificateur de propriété ou de procédure (tels que **Public** ou **Shared**)
- **Modificateur facultatif.** Vous n'êtes pas tenu d'utiliser le modificateur **Overloads** lorsque vous définissez plusieurs propriétés ou procédures surchargées dans la même classe. Toutefois, si vous utilisez **Overloads** dans l'une des déclarations, vous devez l'utiliser dans toutes.
- **L'occultation et la surcharge** de **Overloads** peuvent également être utilisées pour occulter un membre existant ou un ensemble de membres surchargés, dans une classe de base. Lorsque vous utilisez **Overloads** de cette façon, vous déclarez la propriété ou la méthode avec le même nom et la même liste de paramètres que le membre de la classe de base et vous ne spécifiez pas le mot clé **Shadows**.

Le modificateur **Overloads** peut être utilisé dans les contextes suivants :

Function, instruction

Operator, instruction

Property, instruction

Sub, instruction

Voir aussi

Tâches

Comment : définir un opérateur de conversion

Référence

Shadows

Concepts

Surcharge de procédure

Types génériques en Visual Basic

Procédures d'opérateur

Overridable

Spécifie qu'une propriété ou une procédure peut être substituée par une propriété ou une procédure de même nom dans une classe dérivée.

Notes

Overridable est le paramètre par défaut d'une propriété ou d'une procédure qui se substitue à une propriété ou une procédure de classe de base.

L'occultation et la substitution redéfinissent toutes les deux un élément hérité, mais les deux approches sont significativement différentes. Pour plus d'informations, consultez Occultation dans Visual Basic.

Un élément qui peut être substitué est parfois appelé élément *virtuel*. S'il peut être substitué, mais ne doit pas l'être, il est parfois également appelé élément *concret*.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Overridable** uniquement dans une propriété ou une instruction de déclaration de procédure.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **Overridable** avec **MustOverride**, **NotOverridable**, ou **Shared** dans la même déclaration. Étant donné qu'un élément de substitution est implicitement substituable, vous ne pouvez pas combiner **Overridable** avec **Overrides**.

Le modificateur **Overridable** peut être utilisé dans les contextes suivants :

Function, instruction (Visual Basic)

Property, instruction

Sub, instruction (Visual Basic)

Voir aussi

Référence

MustOverride
NotOverridable
Overrides
Mots clés du langage Visual Basic

Concepts

Occultation dans Visual Basic

Overrides

Spécifie qu'une propriété ou une procédure substitue une propriété ou une procédure du même nom héritée d'une classe de base.

Notes

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Overrides** uniquement dans une propriété ou une instruction de déclaration de procédure.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **Overrides** avec **Shadows** ou **Shared** dans la même déclaration. Étant donné qu'un élément de substitution est implicitement substituable, vous ne pouvez pas combiner **Overridable** avec **Overrides**.
- **Correspondance entre signatures.** La signature de cette déclaration doit correspondre exactement à la *signature* de la propriété ou de la procédure qu'elle substitue. Cela signifie que les listes de paramètres doivent avoir le même nombre de paramètres, dans le même ordre, avec les mêmes types de données.

  La déclaration de substitution doit non seulement correspondre à la signature, mais aussi exactement aux éléments suivants :

  - Niveau d'accès
  - Type de retour, le cas échéant
- **Signatures génériques** Dans une procédure générique, la signature inclut le nombre de paramètres de type. Par conséquent, la déclaration de substitution doit correspondre à la version de la classe de base.
- **Correspondance supplémentaire.** Cette déclaration doit non seulement correspondre à la signature de la version de la classe de base, mais aussi lui correspondre dans les cas suivants :
  - Modificateur de niveau d'accès (tel que Public (Visual Basic))
  - Mécanisme de passage de chaque paramètre (ByVal ou ByRef)
  - Listes de contraintes pour chaque paramètre de type d'une procédure générique
- **Occultation et substitution.** L'occultation et la substitution redéfinissent toutes les deux un élément hérité, mais les deux approches sont significativement différentes. Pour plus d'informations, consultez Occultation dans Visual Basic.

Le modificateur **Overrides** peut être utilisé dans les contextes suivants :

Function, instruction

Property, instruction

Sub, instruction

Voir aussi

Référence

MustOverride
NotOverridable
Overridable
Mots clés du langage Visual Basic
Liste de types

Concepts

Occultation dans Visual Basic
Types génériques en Visual Basic

Mots clés P-Z

Le tableau suivant énumère les mots clés du langage Visual Basic.

ParamArray	Partial	Preserve	Private
Protected	Public	RaiseEvent	ReadOnly
Resume	Shadows	Shared	Static
Step	Structure	Text	Then
To	True	TryCast	Unicode
Until	When	While	Widening
WithEvents	WriteOnly		

Notes du développeur sur Smart Device

L'élément suivant n'est pas pris en charge dans les applications Smart Device : **Unicode**.

Voir aussi

Référence

Mots clés A-E

Mots clés F-O

Autres ressources

Référence Visual Basic

ParamArray

Spécifie qu'un paramètre de procédure prend un tableau facultatif d'éléments du type spécifié. **ParamArray** peut être utilisé uniquement sur le dernier paramètre d'une liste de paramètres.

Notes

ParamArray vous permet de passer à la procédure un nombre arbitraire d'arguments. Un paramètre **ParamArray** est toujours déclaré à l'aide de ByVal.

Vous pouvez fournir un ou plusieurs arguments à un paramètre **ParamArray** en passant un tableau du type de données approprié, une liste avec la virgule comme séparateur de valeurs, ou rien du tout. Pour plus d'informations, consultez la section « Appel d'un ParamArray » de la rubrique Tableaux de paramètres.

Remarque de sécurité
Si vous travaillez dans un tableau dont la taille peut être indéfinie, vous risquez de saturer la capacité interne de votre application. Si vous acceptez un tableau de paramètres à partir du code appelant, vous devez tester sa longueur et prendre des mesures appropriées s'il est trop grand pour votre application.

Le modificateur **ParamArray** peut être utilisé dans les contextes suivants :

Declare, instruction

Function, instruction (Visual Basic)

Property, instruction

Sub, instruction (Visual Basic)

Voir aussi

Tâches

Comment : définir une procédure avec un nombre indéfini de paramètres

Comment : appeler une procédure qui accepte un nombre indéfini de paramètres

Référence

Mots clés du langage Visual Basic

Concepts

Tableaux de paramètres

Partial (Visual Basic)

Indique qu'une classe ou une déclaration de structure est une définition partielle de la classe ou de la structure.

```
[ <attrlist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable ] _
Partial { Class | Structure } name [ (Of typelist) ]
    [ Inherits classname ]
    [ Implements interfacenames ]
    [ variabledeclarations ]
    [ proceduredeclarations ]
{ End Class | End Structure }
```

Éléments

attrlist

Facultatif. Liste des attributs qui s'appliquent à cette classe ou structure. Vous devez placer la Liste d'attributs entre signes "inférieur à" et "supérieur à" (**< >**).

accessmodifier

Facultatif. Spécifie le code pouvant accéder à cette classe ou structure. Consultez Niveaux d'accès dans Visual Basic.

Shadows

Facultatif. Consultez Shadows.

MustInherit

Facultatif. Consultez MustInherit.

NotInheritable

Facultatif. Consultez NotInheritable.

name

Requis. Nom de cette classe ou structure. Doit correspondre au nom défini dans toutes les autres déclarations partielles de la même classe ou structure.

Of

Facultatif. Spécifie qu'il s'agit d'une classe ou structure générique. Consultez Types génériques en Visual Basic.

typelist

Requis si vous utilisez De. Consultez Liste de types.

Inherits

Facultatif. Consultez Inherits, instruction.

classname

Requis si vous utilisez **Inherits**. Nom de la classe ou interface dont cette classe dérive.

Implements

Facultatif. Consultez Implements, instruction.

interfacenames

Requis si vous utilisez **Implements**. Noms des interfaces que cette classe ou structure implémente.

variabledeclarations

Facultatif. Instructions qui déclarent des variables et des événements supplémentaires pour la classe ou la structure.

proceduredeclarations

Facultatif. Instructions qui déclarent et définissent des procédures supplémentaires pour la classe ou la structure.

End Class ou **End Structure**

Termine la définition partielle de **Class** de **Structure**.

Notes

Vous pouvez diviser la définition d'une classe ou d'une structure en plusieurs déclarations avec le mot clé **Partial**. Vous pouvez utiliser autant de déclarations partielles que vous le souhaitez, dans autant de fichiers sources différents que vous le souhaitez. Toutefois, toutes les déclarations doivent se trouver dans le même assembly et même espace de noms.

Remarque
Visual Basic utilise des définitions de classe partielle pour séparer le code généré du code utilisateur créé dans des fichiers so urces séparés. Par exemple, le **Concepteur Windows Form** définit des classes partielles pour les contrôles, tels que Form. V ous ne devez pas modifier le code généré dans ces contrôles.

Sous des circonstances normales, vous ne devez pas fractionner le développement d'une classe ou structure unique sur deux déclarations ou plus. Par conséquent, vous n'avez pas besoin du mot clé **Partial** dans la plupart des cas.

Toutes les règles pour la création de classe et de structure, telles que celles pour l'utilisation et l'héritage de modificateur, s'appliquent lors de la création d'une classe ou d'une structure partielle.

Méthodes conseillées

- **Utilisation de mot clé.** Pour la lisibilité, chaque déclaration partielle d'une classe ou structure doit inclure le mot clé **Partial**. Le compilateur permet au plus à une déclaration partielle d'omettre le mot clé ; si deux ou plus l'omettent le compilateur signale une erreur.

Comportement

- **Union de déclarations.** Le compilateur traite la classe ou structure comme l'union de toutes ses déclarations partielles. Chaque modificateur de chaque définition partielle s'applique à la classe ou structure entière et chaque membre de chaque définition partielle est disponible pour la classe ou structure entière.
- **Promotion de type non autorisée pour les types partiels dans les modules.** Si une définition partielle se trouve dans un module, la promotion de type de cette classe ou structure est automatiquement annulée. Dans un tel cas, un jeu de définitions partielles peut provoquer des résultats inattendus ainsi que des erreurs du compilateur. Pour plus d'informations, consultez Promotion de type.

  Le compilateur fusionne des définitions partielles uniquement lorsque leurs chemins qualifiés complets sont identiques.

Le mot clé **Partial** peut être utilisé dans les contextes suivants :

Class, instruction

Structure, instruction

Exemple

L'exemple suivant fractionne la définition de `sampleClass` de classe en deux déclarations chacune d'elles définit une procédure **Sub** différente.

VB

```
Partial Public Class sampleClass
    Public Sub sub1()
    End Sub
End Class
Partial Public Class sampleClass
    Public Sub sub2()
    End Sub
End Class
```

Les deux définitions partielles dans l'exemple précédent pourraient se trouver dans le même fichier source ou dans deux fichiers sources différents.

Voir aussi

Référence

Class, instruction (Visual Basic)
Structure, instruction
Shadows

Concepts

Promotion de type
Types génériques en Visual Basic

Preserve

Empêche l'effacement du contenu d'un tableau lors du redimensionnement de ce dernier.

Notes

Le mot clé **Preserve** peut être utilisé dans le contexte suivant :

ReDim, instruction (Visual Basic)

Voir aussi

Référence

Mots clés du langage Visual Basic

Private (Visual Basic)

Spécifie qu'un ou plusieurs éléments de programmation déclarés sont uniquement accessibles à partir de leur contexte de déclaration, y compris à partir des types contenus.

Notes

Si un élément de programmation représente des fonctionnalités exclusives ou contient des données confidentielles, vous souhaitez généralement limiter son accès aussi strictement que possible. Vous obtenez une limitation maximale en autorisant uniquement le module, la classe ou la structure qui le définit à y avoir accès. Pour limiter l'accès à un élément de cette manière, vous pouvez le déclarer avec **Private**.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Private** seulement au niveau du module. Cela signifie que le contexte de déclaration pour un élément **Private** doit être un module, une classe ou une structure et ne peut pas être un fichier source, un espace de noms ou une procédure.

Comportement

- **Niveau d'accès.** Tout le code d'un contexte de déclaration peut accéder à ses éléments. **Private**. Cela inclut le code dans un type contenu, tel qu'une classe imbriquée ou une expression d'assignation dans une énumération. Aucun code à l'extérieur du contexte de déclaration ne peut accéder à ses éléments **Private**.
- **Modificateurs d'accès.** Les mots clés qui spécifient le niveau d'accès portent le nom de *modificateurs d'accès*. Pour obtenir une comparaison entre les modificateurs d'accès, consultez Niveaux d'accès dans Visual Basic.

Le modificateur **Private** peut être utilisé dans les contextes suivants :

Class, instruction

Const, instruction

Declare, instruction

Delegate, instruction

Dim, instruction

Enum, instruction

Event, instruction

Function, instruction

Interface, instruction

Property, instruction

Structure, instruction

Sub, instruction

Voir aussi

Référence

Public (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)

Concepts

Niveaux d'accès dans Visual Basic
Procédures dans Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données
Présentation des classes

Protected (Visual Basic)

Spécifie que les éléments de programmation déclarés sont uniquement accessibles à partir de leur propre classe ou d'une classe dérivée.

Notes

Parfois un élément de programmation déclaré dans une classe contient des données sensibles ou du code restreint et vous souhaitez limiter l'accès à l'élément. Toutefois, si la classe peut être héritée et si vous anticipez une hiérarchie de classes dérivées, il peut s'avérer nécessaire pour ces classes dérivées d'accéder aux données ou au code. Dans ce cas, vous souhaitez que l'élément soit à la fois accessible de la classe de base et de toutes les classes dérivées. Pour limiter l'accès à un élément de cette manière, vous pouvez le déclarer avec **Protected**.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Protected** seulement au niveau de la classe. Cela signifie que le contexte de déclaration pour un élément **Protected** doit être une classe et ne peut pas être un fichier source, un espace de noms, une interface, un module, une structure ou une procédure.
- **Modificateurs combinés.** Vous pouvez utiliser le modificateur **Protected** conjointement avec le modificateur Friend (Visual Basic) dans la même déclaration. Cette combinaison rend les éléments déclarés accessibles de partout dans le même assembly, de leur propre classe, et de classes dérivées. Vous pouvez spécifier **Protected Friend** uniquement sur les membres d'une classe.

Comportement

- **Niveau d'accès.** Tout le code dans une classe peut accéder à ses éléments. Le code dans toute classe qui dérive d'une classe de base peut accéder à tous les éléments **Protected** de la classe de base. Cette règle est valable pour toutes les générations de dérivation, ce qui signifie qu'une classe peut accéder aux éléments **Protected** de la classe de base de la classe de base, et ainsi de suite.

  L'accès protégé n'est pas un sur-ensemble ou sub-set de l'accès ami.
- **Modificateurs d'accès.** Les mots clés qui spécifient le niveau d'accès portent le nom de *modificateurs d'accès*. Pour obtenir une comparaison entre les modificateurs d'accès, consultez Niveaux d'accès dans Visual Basic.

Le modificateur **Protected** peut être utilisé dans les contextes suivants :

Class, instruction

Const, instruction

Declare, instruction

Delegate, instruction

Dim, instruction

Enum, instruction

Event, instruction

Function, instruction

Interface, instruction

Property, instruction

Structure, instruction

Sub, instruction

Voir aussi

Référence

Public (Visual Basic)
Friend (Visual Basic)
Private (Visual Basic)

Concepts

Niveaux d'accès dans Visual Basic

Procédures dans Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données

Présentation des classes

Public (Visual Basic)

Spécifie que certains éléments de programmation déclarés ne présentent aucune limitation d'accès.

Notes

Vous publiez généralement un ou plusieurs composants, tels qu'une bibliothèque de classes, afin que les éléments de programmation soient accessibles par n'importe quel code qui interagit avec votre assembly. Pour octroyer cet accès illimité à un élément, vous pouvez le déclarer avec **Public**.

L'accès public représente le niveau normal pour un élément de programmation lorsque vous ne devez pas limiter son accès. Notez que le niveau d'accès d'un élément déclaré dans une interface, un module, une classe ou une structure a la valeur par défaut **Public** si vous n'affectez pas une autre valeur.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Public** seulement au niveau du module, de l'interface ou de l'espace de noms. Cela signifie que le contexte de déclaration pour un élément **Public** doit être un fichier source, un espace de noms, un module, une classe ou une structure, et ne peut pas être une procédure.

Comportement

- **Niveau d'accès.** Si le code peut accéder à un module, une classe ou une structure, il peut accéder à ses éléments **Public**.
- **Accès par défaut.** Variables locales dans une procédure à accès public par défaut sur lesquelles vous ne pouvez pas utiliser de modificateurs d'accès.
- **Modificateurs d'accès.** Les mots clés qui spécifient le niveau d'accès portent le nom de *modificateurs d'accès*. Pour obtenir une comparaison entre les modificateurs d'accès, consultez Niveaux d'accès dans Visual Basic.

Le modificateur **Public** peut être utilisé dans les contextes suivants :

Class, instruction

Const, instruction

Declare, instruction

Delegate, instruction

Dim, instruction

Enum, instruction

Event, instruction

Function, instruction

Interface, instruction

Module, instruction

Operator, instruction

Property, instruction

Structure, instruction

Sub, instruction

Voir aussi

Référence

Protected (Visual Basic)
Friend (Visual Basic)
Private (Visual Basic)

Concepts

Niveaux d'accès dans Visual Basic
Procédures dans Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données

Présentation des classes

RaiseEvent

Déclenche un événement au moment de l'exécution, ou déclare le code à exécuter lors du déclenchement d'un événement.

Notes

Le mot clé **RaiseEvent** peut être utilisé dans les contextes suivants :

Event, instruction

Le mot clé **RaiseEvent** déclare un accesseur **RaiseEvent** personnalisé.

RaiseEvent, instruction

Le mot clé **RaiseEvent** déclenche un événement.

Voir aussi

Tâches

Comment : déclencher un événement (Visual Basic)
Comment : déclarer des événements qui évitent les blocages
Comment : déclarer des événements qui économisent l'utilisation de la mémoire

ReadOnly (Visual Basic)

Spécifie qu'une variable ou une propriété peut être lue, mais pas écrite.

Notes
Règles

- **Contexte de déclaration.** Vous pouvez utiliser **ReadOnly** seulement au niveau du module. Cela signifie que le contexte de déclaration pour un élément **ReadOnly** doit être une classe, une structure ou un module, et ne peut pas être un fichier source, un espace de noms ou une procédure.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **ReadOnly** avec **Static** dans la même déclaration.
- **Assignation d'une valeur.** Le code utilisant une propriété **ReadOnly** ne peut pas définir sa valeur. Mais le code qui a accès au stockage sous-jacent peut assigner ou modifier la valeur n'importe quand.

  Vous pouvez assigner une valeur à une variable **ReadOnly** uniquement dans sa déclaration ou dans le constructeur d'une classe ou d'une structure dans lequel elle est définie.

Quand utiliser une variable en lecture seule

Il y a des situations dans lesquelles vous ne pouvez pas utiliser un Const, instruction (Visual Basic) pour déclarer et assigner une valeur de constante. Par exemple, l'instruction **Const** ne pourra peut-être pas accepter le type de données que vous souhaitez assigner ou vous pouvez ne pas être en mesure de calculer la valeur au moment de la compilation avec une expression constante. Il est même possible que vous ne connaissiez pas la valeur au moment de la compilation. Dans ce cas, vous pouvez utiliser une variable **ReadOnly** pour contenir une valeur de constante.

Remarque de sécurité
Si le type de données de la variable est un type référence, tel qu'un tableau ou une instance de classe, ses membres peuvent être changés même si la variable elle-même est **ReadOnly**. L'exemple suivant illustre ce comportement :

```
ReadOnly characterArray() As Char = {"x"c, "y"c, "z"c}

Sub changeArrayElement()

characterArray(1) = "M"c

End Sub
```

Lorsqu'il est initialisé, le tableau vers lequel pointe `characterArray()` contient "x", "y" et "z". Dans la mesure où la variable `characterArray` est **ReadOnly**, vous ne pouvez pas modifier sa valeur une fois qu'elle est initialisée, c'est-à-dire que vous ne pouvez pas lui assigner un nouveau tableau. Toutefois, vous pouvez modifier les valeurs d'un ou plusieurs des membres du tableau. Suite à un appel à `changeArrayElement` de la procédure, le tableau vers lequel pointe `characterArray()` contient "x", "M" et "z".

Notez que ce procédé est semblable à la déclaration d'un paramètre d'une procédure en tant que ByVal, ce qui empêche la procédure de modifier l'argument appelant lui-même, mais lui permet de modifier ses membres.

Exemple

L'exemple suivant définit une propriété **ReadOnly** de la date à laquelle un employé a été embauché. La classe stocke en interne la valeur de propriété en tant que variable **Private** et seul le code à l'intérieur de la classe peut modifier cette valeur. Toutefois, la propriété est **Public** et tout code qui peut accéder à la classe peut lire la propriété.

VB

```
Class employee
    ' Only code inside class employee can change the value of hireDateValue.
    Private hireDateValue As Date
    ' Any code that can access class employee can read property dateHired.
    Public ReadOnly Property dateHired() As Date
        Get
            Return hireDateValue
        End Get
    End Property
End Class
```

Le modificateur **ReadOnly** peut être utilisé dans les contextes suivants :

Dim, instruction

Property, instruction

Voir aussi
Référence
WriteOnly
Mots clés du langage Visual Basic

RemoveHandler

Supprime l'association entre un événement et un gestionnaire d'événements, ou déclare le code à exécuter lors de l'ajout d'un gestionnaire d'événements.

Notes

Le mot clé **RemoveHandler** peut être utilisé dans les contextes suivants :

Event, instruction

Le mot clé **RemoveHandler** déclare un accesseur **RemoveHandler** personnalisé.

RemoveHandler, instruction

Le mot clé **RemoveHandler** supprime l'association qui lie un événement et un gestionnaire d'événements.

Voir aussi

Tâches

Comment : déclarer des événements qui évitent les blocages

Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Resume

Spécifie l'endroit où doit se poursuivre l'exécution après qu'une erreur a été gérée.

Notes

Le mot clé **Resume** peut être utilisé dans les contextes suivants :

On Error, instruction (Visual Basic)

Resume, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

Shadows

Spécifie qu'un élément de programmation déclaré redéclare et masque un élément de programmation de même nom ou définit un ensemble d'éléments surchargés, dans une classe de base.

Notes

Le principal objectif de l'occultation (qui est également appelée *masquage par nom*) est de conserver la définition de vos membres de classe. La classe de base peut subir un changement qui crée un élément du même nom que celui que vous avez déjà défini. Dans ce cas, le modificateur **Shadows** oblige les références de votre classe à être résolues vers le membre que vous avez défini, et non vers le nouvel élément de la classe de base.

L'occultation et la substitution redéfinissent toutes les deux un élément hérité, mais les deux approches sont significativement différentes. Pour plus d'informations, consultez Occultation dans Visual Basic.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Shadows** seulement au niveau de la classe. Cela signifie que le contexte de déclaration pour un élément **Shadows** doit être une classe et ne peut pas être un fichier source, un espace de noms, une interface, un module, une structure ou une procédure.

  Vous ne pouvez déclarer qu'un élément d'occultation dans une seule instruction de déclaration.

- **Modificateurs combinés.** Vous ne pouvez pas spécifier **Shadows** avec **Overloads**, **Overrides**, ou **Static** dans la même déclaration.

- **Types d'éléments.** Vous pouvez occulter tout type d'élément déclaré par un autre type. Si vous masquez une propriété ou une procédure avec une autre propriété ou procédure, les paramètres et le type retourné n'ont pas besoin de correspondre à ceux de la propriété ou de la procédure de la classe de base.

- **Accès.** L'élément occulté dans la classe de base n'est normalement pas disponible dans la classe dérivée qui l'occulte. Cependant, les considérations suivantes s'appliquent :

  - Si l'élément d'occultation n'est pas disponible à partir du code qui s'y réfère, la référence correspond à l'élément occulté. Par exemple, si un élément **Private** occulte un élément de classe de base, le code qui n'est pas autorisé à accéder à l'élément **Private** accède à la place à l'élément de classe de base.

  - Si vous occultez un élément, vous pouvez toujours accéder à l'élément occulté par l'intermédiaire d'un objet déclaré avec le type de classe de base. Vous pouvez également y accéder par l'intermédiaire de **MyBase**.

Le modificateur **Shadows** peut être utilisé dans les contextes suivants :

Class, instruction

Const, instruction

Declare, instruction

Delegate, instruction

Dim, instruction

Enum, instruction

Event, instruction

Function, instruction

Interface, instruction

Property, instruction

Structure, instruction

Sub, instruction

Voir aussi

Référence

Shared (Visual Basic)

Static (Visual Basic)

Private (Visual Basic)
MyBase
MustOverride
NotOverridable
Overloads
Overridable
Overrides

Concepts

Occultation dans Visual Basic

Shared (Visual Basic)

Spécifie que les éléments de programmation déclarés sont associés à une classe ou structure, et non à une instance spécifique de la classe ou structure.

Notes

Quand utiliser Shared

Le partage d'un membre d'une classe ou structure le rend disponible à chaque instance, contrairement à *non partagé* où chaque instance conserve sa propre copie. Par exemple, le fait que la valeur d'une variable s'applique à l'application entière est utile. Si vous déclarez la variable comme étant **Shared**, toutes les instances accéderont au même emplacement de stockage, et si une instance modifie la valeur de la variable, toutes les instances accèdent à la valeur mise à jour.

Le partage n'altère pas le niveau d'accès d'un membre. Par exemple, un membre de classe peut être partagé et privé (accessible uniquement à partir de la classe), ou non partagé et public. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser **Shared** seulement au niveau du module. Cela signifie que le contexte de déclaration d'un élément **Shared** doit être une classe ou une structure, et ne peut pas être un fichier source, un espace de noms, ou une procédure.
- **Modificateurs combinés.** Vous ne pouvez pas spécifier **Shared** avec Overrides, Overridable, NotOverridable, MustOverride, ou Static (Visual Basic) dans la même déclaration.
- **Accès.** Vous pouvez accéder à un élément partagé en le qualifiant avec son nom de classe ou de structure, non avec le nom de variable d'une instance spécifique de sa classe ou structure. Vous n'êtes même pas obligé de créer une instance d'une classe ou structure pour accéder à ses membres partagés.

  L'exemple suivant appelle la procédure IsNaN partagée exposée par la structure Double.

  ```
  If Double.IsNaN(result) Then MsgBox("Result is mathematically undefined.")
  ```

- **Partage implicite.** Vous ne pouvez pas utiliser le modificateur **Shared** dans un Const, instruction (Visual Basic), mais les constantes sont partagées implicitement. De la même façon, vous ne pouvez pas déclarer un membre de module ou d'interface comme étant **Shared**, mais ils sont partagés implicitement.

Comportement

- **Stockage.** Une variable ou un événement partagé est stocké une seule fois dans la mémoire, sans tenir compte du nombre d'instances de sa classe ou structure que vous créez. De la même façon, une procédure ou une propriété partagée contient seulement un jeu de variables locales.
- **Accès via une variable d'instance.** Il est possible d'accéder à un élément partagé en le qualifiant avec le nom d'une variable qui contient une instance spécifique de sa classe ou structure. Bien que cette technique fonctionne normalement, le compilateur génère un message d'avertissement et autorise l'accès via la classe ou le nom de structure au lieu de la variable.
- **Accès via une expression d'instance.** Si vous accédez à un élément partagé via une expression qui retourne une instance de sa classe ou structure, le compilateur autorise l'accès via la classe ou le nom de structure au lieu d'évaluer l'expression. Des résultats inattendus se produisent, autres que l'exécution par l'expression d'autres actions ainsi que le retour de l'instance. L'exemple suivant illustre ce comportement :

```
Sub main()
    shareTotal.total = 10
    ' The preceding line is the preferred way to access total.
    Dim instanceVar As New shareTotal
    instanceVar.total += 100
    ' The preceding line generates a compiler warning message and
    ' accesses total through class shareTotal instead of through
    ' the variable instanceVar. This works as expected and adds
    ' 100 to total.
    returnClass().total += 1000
```

```
    ' The preceding line generates a compiler warning message and
    ' accesses total through class shareTotal instead of calling
    ' returnClass(). This adds 1000 to total but does not work as
    ' expected, because the MsgBox in returnClass() does not run.
    MsgBox("Value of total is " & CStr(shareTotal.total))
End Sub
Public Function returnClass() As shareTotal
    MsgBox("Function returnClass() called")
    Return New shareTotal
End Function
Public Class shareTotal
    Public Shared total As Integer
End Class
```

Dans l'exemple précédent, le compilateur génère un message d'avertissement les deux fois que le code accède à la variable partagée `total` via une instance. Dans chaque cas, il autorise un accès direct via le `shareTotal` de classe et n'utilise aucune instance. Dans le cas où l'appel prévu au `returnClass` de la procédure se produit, il ne génère même pas d'appel à `returnClass`, donc l'action supplémentaire d'affichage de "Function returnClass() called" n'est pas exécutée.

Le modificateur **Shared** peut être utilisé dans les contextes suivants :

Dim, instruction

Event, instruction

Function, instruction

Operator, instruction

Property, instruction

Sub, instruction

Voir aussi

Référence

Shadows

Static (Visual Basic)

Concepts

Durée de vie dans Visual Basic

Procédures dans Visual Basic

Membres partagés en Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données

Présentation des classes

Static (Visual Basic)

Spécifie qu'une ou plusieurs variables locales déclarées doivent continuer à exister et conservent leurs valeurs les plus récentes après l'arrêt de la procédure dans laquelle elles sont déclarées.

Notes

Normalement, une variable locale dans une procédure cesse d'exister dès que la procédure se termine. Une variable statique continue à exister et conserve sa valeur la plus récente. La prochaine fois que votre code appellera la procédure, la variable ne sera pas réinitialisée et elle contiendra encore la valeur la plus récente que vous lui aviez assignée. Une variable statique continue à exister pendant toute la durée de vie de la classe ou du module dans lequel elle est définie.

Règles

- **Contexte de déclaration.** Vous pouvez utiliser uniquement **Static** sur les variables locales. Cela signifie que le contexte de déclaration pour une variable **Static** doit être une procédure ou un bloc dans une procédure et ne peut pas être un fichier source, un espace de noms, une classe, une structure ou un module.

  Vous ne pouvez pas utiliser **Static** à l'intérieur d'une procédure de structure.

- **Modificateurs combinés.** Vous ne pouvez pas spécifier **Static** avec **ReadOnly**, **Shadows**, ou **Shared** dans la même déclaration.

Comportement

Le comportement de toute variable locale est différent suivant qu'elle soit déclarée dans une procédure **Shared** ou non. Si la procédure est **Shared**, toutes ses variables locales sont partagées automatiquement, y compris les variables **Static**. Il y a uniquement une copie d'une variable de ce type pour l'application entière. Vous appelez une procédure **Shared** à l'aide du nom de classe, et non d'un pointage variable vers une instance de la classe.

Si la procédure n'est pas **Shared**, ses variables locales sont des *variables d'instance*, y compris les variables **Static**. Il y a une copie indépendante de chaque variable dans chaque instance de la classe. Vous appelez une procédure non partagée à l'aide d'un pointage variable vers une instance spécifique de la classe. Toute variable dans cette instance est indépendante d'une variable avec le même nom dans une autre instance. Par conséquent, elles peuvent maintenir des valeurs différentes.

Exemple

L'exemple suivant illustre l'utilisation du mot clé **Static** :

VB

```
Function updateSales(ByVal thisSale As Decimal) As Decimal
    Static totalSales As Decimal = 0
    totalSales += thisSale
    Return totalSales
End Function
```

La variable **Static** variable `totalSales` est initialisée une seule fois sur 0. Chaque fois que vous entrez `updateSales`, `totalSales` a encore la valeur la plus récente que vous avez calculé pour lui.

Le modificateur **Static** peut être utilisé dans le contexte suivant :

Dim, instruction (Visual Basic)

Voir aussi

Référence

Shadows

Shared (Visual Basic)

Concepts

Durée de vie dans Visual Basic

Déclaration de variable dans Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données

Présentation des classes

Step

Spécifie une valeur d'incrément pour un compteur de boucle.

Notes

Le mot clé **Step** peut être utilisé dans le contexte suivant :

For...Next, instruction (Visual Basic)

Voir aussi

Référence

Mots clés du langage Visual Basic

Structure (Visual Basic)

Contraint un paramètre de type générique à demander que tous les arguments de type qui lui sont passés soient des types valeur, ou introduit une instruction Structure.

Notes

Lorsque vous déclarez un paramètre de type pour un type générique, vous pouvez imposer une *contrainte*, un jeu d'un ou plusieurs conditions requises qui limitent l'argument de type qui peut être passé à ce paramètre de type. Pour plus d'informations, consultez « Constraints » dans Types génériques en Visual Basic.

Une des conditions requises possibles dans une contrainte est que l'argument de type doit être un type valeur, par exemple une structure, une énumération ou un type de données élémentaire. Cette spécification s'effectue en incluant le mot clé **Structure** dans la contrainte.

Une autre condition requise possible est que l'argument de type doit être un type référence, par exemple un **String**, un tableau ou un délégué, ou un objet créé à partir d'une classe. Vous spécifiez une condition requise de type référence en incluant le mot clé Class (Visual Basic) dans la contrainte.

Vous ne devez spécifier ni **Class** ni **Structure** dans une contrainte. Vous ne pouvez pas les spécifier à la fois dans la même contrainte.

La contrainte **Structure** est différente de l'instruction Structure.

Voir aussi

Référence

Class (Visual Basic)

Structure, instruction

Concepts

Types génériques en Visual Basic

Types valeur et types référence

Text

Définit la méthode de comparaison des chaînes pour trier les caractères de texte équivalents.

Notes

L'instruction **Option Compare** doit apparaître dans un fichier source avant toute autre instruction. Elle affecte la comparaison des chaînes pour tout le code généré par ce fichier source.

Quand utiliser la propriété Text

Si vos chaînes contiennent tous les caractères de texte, il vous arrivera souvent de vouloir les comparer en prenant en considération les équivalences alphabétiques telles que le non-respect de la casse et les caractères collés. Par exemple, il se peut que vous souhaitiez considérer `A` et `a` comme deux caractères équivalents et `Ä` et `ä` comme deux caractères précédant `B` et `b`. Pour spécifier une comparaison de texte, définissez **Option Compare** sur **Text**.

Le mot clé **Text** peut être utilisé dans le contexte suivant :

Option Compare, instruction

Voir aussi
Référence
Binary
Mots clés du langage Visual Basic

Then

Introduit un bloc d'instructions à compiler ou exécuter si une condition testée est true.

Notes

Le mot clé **Then** peut être utilisé dans les contextes suivants :

Directive #If...Then...#Else

Instruction If...Then...Else

Voir aussi

Référence

Mots clés du langage Visual Basic

To

Sépare les valeurs et début et de fin d'un compteur de boucle ou d'une plage de valeurs correspondante.

Notes

Le mot clé **To** peut être utilisé dans les contextes suivants :

For...Next, instructions

Select...Case, instruction

Voir aussi

Référence

Mots clés du langage Visual Basic

True (Visual Basic)

Représente une valeur **Boolean** qui passe un test conditionnel.

Notes

Les valeurs Boolean, type de données (Visual Basic) ne sont pas stockées en tant que nombres et les valeurs stockées ne sont pas conçues pour être équivalentes aux nombres. Vous ne devez jamais écrire un code dépend de valeurs numériques équivalentes à **True** et **False**. Autant que possible, limitez l'utilisation des variables **Boolean** aux valeurs logiques pour lesquelles elles sont conçues. S'il s'avère nécessaire de mélanger des valeurs **Boolean** et des valeurs numériques, assurez-vous de comprendre la méthode de conversion choisie. Pour plus d'informations, consultez « Boolean Type Does Not Convert to Numeric Type Accurately » dans Dépannage des types de données.

Voir aussi

Tâches

Dépannage des types de données

Référence

Boolean, type de données (Visual Basic)

False (Visual Basic)

TryCast

Introduit une opération de conversion de type qui ne lève pas d'exception.

Notes

Si une tentative de conversion échoue, **CType** et **DirectCast** lèvent tous deux une erreur InvalidCastException. Cela peut nuire à la performance de votre application. **TryCast** retourne Nothing (Visual Basic), de telle façon qu'au lieu d'avoir à gérer une éventuelle exception, vous n'aurez qu'à tester le résultat obtenu pour **Nothing**.

Utilisez le mot clé **TryCast** de la même façon que si vous utilisiez les mots clés CType ou DirectCast. Vous fournissez comme premier argument une expression et comme deuxième argument le type cible de la conversion. **TryCast** fonctionne uniquement sur des types référence, tels que des classes et des interfaces. Une relation d'héritage ou d'implémentation entre les deux types est requise. Cela signifie qu'un type doit hériter de l'autre ou l'implémenter.

Erreurs et échecs

TryCast génère une erreur du compilateur s'il détecte qu'il n'y a aucune relation d'héritage ou d'implémentation. Cependant, l'absence d'erreur du compilateur ne garantit pas une conversion réussie. Si la conversion souhaitée est restrictive, elle peut échouer au moment de l'exécution. Le cas échéant, **TryCast** retourne Nothing (Visual Basic).

Mots clés de conversion

Une comparaison des mots clés de conversion de type se présente comme suit.

Mot clé	Types de données	Relation d'argument	Échec lors de l'exécution
CType, fonction	Tout type de données	Une conversion étendue ou restrictive doit être définie entre les deux types de données	Lève **InvalidCastException**
DirectCast	Tout type de données	Un type doit hériter de l'autre type ou l'implémenter	Lève **InvalidCastException**
TryCast	Types référence uniquement	Un type doit hériter de l'autre type ou l'implémenter	Retourne Nothing (Visual Basic)

Exemple

L'exemple suivant montre comment utiliser **TryCast**.

VB

```
Function PrintTypeCode(ByVal obj As Object) As String
    Dim objAsConvertible As IConvertible = TryCast(obj, IConvertible)
    If objAsConvertible Is Nothing Then
        Return obj.ToString() & " does not implement IConvertible"
    Else
        Return "Type code is " & objAsConvertible.GetTypeCode()
    End If
End Function
```

Voir aussi

Concepts

Conversions étendues et restrictives

Conversions implicites et explicites

Unicode (Visual Basic)

Spécifie que Visual Basic doit marshaler toutes les chaînes en valeurs Unicode indépendamment du nom de la procédure externe qui est déclarée.

Lorsque vous appelez une procédure définie à l'extérieur de votre projet, le compilateur Visual Basic n'a pas accès aux informations requises pour appeler la procédure correctement. Ces informations incluent l'emplacement de la procédure, son identification, sa séquence d'appel et son type de retour ainsi que le jeu de caractères de chaîne qu'elle utilise. L'instruction Declare crée une référence à une procédure externe et fournit ces informations nécessaires.

L'élément *charsetmodifier* de l'instruction **Declare** fournit les informations de jeu de caractères pour marshaler des chaînes lors d'un appel à la procédure externe. Il affecte également la façon dont Visual Basic recherche le nom de la procédure externe dans le fichier externe. Le modificateur **Unicode** spécifie que Visual Basic doit marshaler toutes les chaînes en valeurs Unicode et doit rechercher la procédure sans modifier son nom lors de la recherche.

Si aucun modificateur de jeu de caractères n'est spécifié, **Ansi** s'applique par défaut.

Notes

Le modificateur **Unicode** peut être utilisé dans le contexte suivant :

Declare, instruction

Notes du développeur sur Smart Device

Ce mot clé n'est pas pris en charge.

Voir aussi

Référence

Ansi
Auto
Mots clés du langage Visual Basic

Until

Spécifie une condition qui termine l'exécution d'une boucle **Do**.

Notes

Le mot clé **Until** peut être utilisé dans le contexte suivant :

Instruction Do...Loop

Voir aussi

Référence

Mots clés du langage Visual Basic

When

Ajoute un test conditionnel à une instruction **Catch**.

Notes

Le mot clé **When** peut être utilisé dans le contexte suivant :

Instruction Try...Catch...Finally

Voir aussi

Référence

Mots clés du langage Visual Basic

While (Visual Basic)

Spécifie une condition qui autorise l'exécution d'une boucle **Do** pour continuer.

Notes

Le mot clé **While** est différent de l'instruction d'ouverture du bloc **While...End While**.

Le mot clé **While** peut être utilisé dans le contexte suivant :

Do...Loop, instruction

Voir aussi

Référence

While...End While, instruction (Visual Basic)

Widening

Indique qu'un opérateur de conversion (**CType**) convertit une classe ou une structure en un type qui peut contenir toutes les valeurs possibles de la classe ou de la structure d'origine.

Conversion avec le mot clé Widening

La procédure de conversion doit spécifier **Public Shared** en plus de **Widening**.

Les conversions étendues réussissent toujours au moment de l'exécution et sans jamais aucune perte de données. Les exemples sont **Single** en **Double**, **Char** en **String** et un type dérivé en un type de base. Cette dernière conversion est étendue parce que le type dérivé contient tous les membres du type de base et qu'il s'agit, par conséquent, d'une instance du type de base.

Le code utilisateur ne doit pas utiliser **CType** pour les conversions étendues, même si **Option Strict** est **On**.

Le mot clé **Widening** peut être utilisé dans le contexte suivant :

Operator, instruction

Voir aussi

Tâches

Comment : définir un opérateur

Référence

Operator, instruction
Narrowing
CType, fonction
Option Strict, instruction

Concepts

Conversions étendues et restrictives

WithEvents

Indique qu'une ou plusieurs variables membres déclarées font référence à une instance d'une classe qui peut déclencher des événements.

Notes

Lorsqu'une variable est définie à l'aide de **WithEvents**, vous pouvez spécifier de façon déclarative qu'une méthode gère les événements de la variable à l'aide du mot clé **Handles**.

Vous pouvez utiliser **WithEvents** seulement au niveau de la classe ou du module. Cela signifie que le contexte de déclaration pour une variable **WithEvents** doit être une classe ou un module, et ne peut pas être un fichier source, un espace de noms, une structure ou une procédure.

Vous ne pouvez pas utiliser **WithEvents** sur un membre de structure.

Vous ne pouvez déclarer que des variables, et non des tableaux, avec **WithEvents**.

Règles

- **Types d'éléments.** Vous devez déclarer les variables **WithEvents** comme des variables objet afin qu'elles puissent accepter des instances de classe. Toutefois, vous ne pouvez pas les déclarer comme **Object**. Vous devez les déclarer comme la classe spécifique capable de déclencher les événements.

Le modificateur **WithEvents** peut être utilisé dans le contexte suivant : Dim, instruction (Visual Basic)

Voir aussi

Référence

Handles

Mots clés du langage Visual Basic

Concepts

WithEvents et la clause Handles

Autres ressources

Événements en Visual Basic

WriteOnly

Spécifie qu'une propriété peut être écrite, mais pas lue.

Notes

Règles

Contexte de déclaration. Vous pouvez utiliser **WriteOnly** seulement au niveau du module. Cela signifie que le contexte de déclaration pour une propriété **WriteOnly** doit être une classe, une structure ou un module, et ne peut pas être un fichier source, un espace de noms ou une procédure.

Vous pouvez déclarer une propriété en tant que **WriteOnly**, mais pas une variable.

Quand utiliser WriteOnly

Parfois vous souhaitez que le code utilisateur soit capable de définir une valeur, mais pas de découvrir ce qu'elle est. Par exemple, les données sensibles, telles qu'un numéro d'immatriculation social ou un mot de passe, doivent être protégé de l'accès par tout composant qui n'a pas défini ces données. Dans ces cas, vous pouvez utiliser une propriété **WriteOnly** pour définir la valeur.

Remarque de sécurité
Lorsque vous définissez et utilisez une propriété **WriteOnly**, tenez compte des mesures de protection supplémentaires suivantes :

- **Substitution.** Si la propriété est un membre d'une classe, sa valeur par défaut doit être NotOverridable et ne la déclarez pas en tant que **Overridable** ou **MustOverride**. Cela empêche une classe dérivée d'accéder de manière indésirable via une substitution.
- **Niveau d'accès.** Si vous stockez les données sensibles de la propriété dans une ou plusieurs variables, déclarez-les en tant que Private (Visual Basic) afin qu'aucun autre code ne puisse y accéder.
- **Chiffrement.** Stockez toutes les données sensibles sous forme chiffrée plutôt qu'en texte brut. Si du code nuisible accède d'une façon ou d'une autre à la zone de mémoire, il est plus difficile d'utiliser les données. Le chiffrement est également utile s'il s'avère nécessaire de sérialiser les données sensibles.
- **Réinitialisation.** Lorsque la classe, la structure ou le module définissant la propriété sont arrêtés, réinitialisez les données sensibles à leurs valeurs par défaut ou à d'autres valeurs sans signification. Cela apporte une protection supplémentaire lorsque cette zone de mémoire est libérée pour l'accès général.
- **Persistance.** Si vous pouvez l'éviter, ne rendez pas persistante toute donnée sensible, par exemple sur un disque. De la même façon, n'écrivez pas de données sensibles dans le Presse-papiers.

Le modificateur **WriteOnly** peut être utilisé dans le contexte suivant :

Property, instruction

Voir aussi

Référence

ReadOnly (Visual Basic)
Private (Visual Basic)
Mots clés du langage Visual Basic

Modules (Visual Basic)

Les rubriques de cette section documentent les modules exécutables Visual Basic et contiennent des tables de leurs procédures, propriétés, constantes et énumérations membres.

Dans cette section

Constants, module (Visual Basic)

ControlChars, module (Visual Basic)

Conversion, module (Visual Basic)

DateAndTime, module (Visual Basic)

ErrObject, module (Visual Basic)

FileSystem, module (Visual Basic)

Financial, module (Visual Basic)

Globals, module (Visual Basic)

Information, module (Visual Basic)

Interaction, module (Visual Basic)

Strings, module (Visual Basic)

VbMath, module (Visual Basic)

Rubriques connexes

Référence Visual Basic

Visual Basic

Constants, module (Visual Basic)

Le module **Constants** contient diverses constantes. Ces constantes peuvent être utilisées partout dans votre code.

Notes

Ce module prend en charge les constantes du langage Visual Basic qui correspondent aux membres de plusieurs énumérations Visual Basic :

Membres

vbAbort	vbAbortRetryIgnore	vbApplicationModal	vbArchive
vbArray	vbBack	vbBinaryCompare	vbBoolean
vbByte	vbCancel	vbCr	vbCritical
vbCrLf	vbCurrency	vbDate	vbDecimal
vbDefaultButton1	vbDefaultButton2	vbDefaultButton3	vbDirectory
vbDouble	vbEmpty	vbExclamation	vbFalse
vbFirstFourDays	vbFirstFullWeek	vbFirstJan1	vbFormFeed
vbFriday	VbGeneralDate	vbGet	vbHidden
vbHide	vbHiragana	vbIgnore	vbInformation
vbInteger	vbKatakana	vbLet	vbLf
vbLinguisticCasing	vbLong	VbLongDate	VbLongTime
vbLowerCase	vbMaximizedFocus	vbMethod	vbMinimizedFocus
vbMinimizedNoFocus	vbMonday	vbMsgBoxHelp	vbMsgBoxRight
vbMsgBoxRtlReading	vbMsgBoxSetForeground	vbNarrow	vbNewLine
vbNo	vbnormal	vbNormalFocus	vbNormalNoFocus
vbNull	vbNullChar	vbNullString	vbObject
vbObjectError	vbOK	vbOKCancel	vbOKOnly
vbProperCase	vbQuestion	vbReadOnly	vbRetry
vbRetryCancel	vbSaturday	vbSet	VbShortDate
VbShortTime	vbSimplifiedChinese	vbSingle	vbString
vbSunday	vbSystem	vbSystemModal	vbTab
vbTextCompare	vbThursday	vbTraditionalChinese	vbTrue

vbTuesday	vbUpperCase	vbUseDefault	vbUserDefinedType
vbUseSystem	vbUseSystemDayOfWeek	vbVariant	vbVerticalTab
vbVolume	vbWednesday	vbWide	vbYes
vbYesNo	vbYesNoCancel		

Voir aussi

Référence

AppWinStyle, énumération
BuiltInRole, énumération
CallType, énumération
CompareMethod, énumération
DateFormat, énumération
DateInterval, énumération
DueDate, énumération
FileAttribute, énumération
FirstDayOfWeek, énumération
FirstWeekOfYear, énumération
MsgBoxResult, énumération
MsgBoxStyle, énumération
OpenAccess, énumération
OpenMode, énumération
OpenShare, énumération
Constantes d'impression et d'affichage
Tristate, énumération
VariantType, énumération
VbStrConv, énumération
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

ControlChars, module (Visual Basic)

Le module **ControlChars** contient les constantes utilisées comme caractères de contrôle. Ces constantes peuvent être utilisées partout dans votre code.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime qui fournissent des caractères de contrôle de constante pour imprimer et afficher le texte. Pour plus d'informations, consultez Constantes d'impression et d'affichage.

Membres

Back	Cr	CrLf	FormFeed
Lf	NewLine	NullChar	Quote
Tab	VerticalTab		

Voir aussi

Référence

Constantes d'impression et d'affichage
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Conversion, module (Visual Basic)

Le module **Conversion** contient les procédures utilisées pour effectuer diverses opérations de conversion.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime qui convertissent des nombres décimaux en d'autres bases, des nombres en chaînes, des chaînes en nombres et un type de données en un autre.

Membres

ErrorToString	Fix	Hex	Int
Oct	Str	Val	

Exemple

Cet exemple utilise la fonction **Hex** pour retourner la valeur hexadécimale d'un nombre.

VB

```
Dim TestHex As String
' Returns 5.
TestHex = Hex(5)
' Returns A.
TestHex = Hex(10)
' Returns 1CB.
TestHex = Hex(459)
```

Voir aussi

Référence

Liste des conversions
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

DateAndTime, module (Visual Basic)

Le module **DateAndTime** contient les procédures et propriétés utilisées pour les opérations de date et d'heure.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime qui obtiennent la date ou l'heure actuelle, effectuent des calculs de date, retournent une date ou une heure, définissent la date ou l'heure ou calculent la durée d'un processus.

Membres

DateAdd	DateDiff	DatePart	DateSerial
DateString	DateValue	Day	Hour
Minute	Month	MonthName	Now
Second	TimeOfDay	Timer	TimeSerial
TimeString	TimeValue	Today	Weekday
WeekDayName	Year		

Exemple

Cet exemple utilise la propriété **Today** pour retourner la date système actuelle.

```
Dim thisDate As Date
thisDate = Today
```

Voir aussi

Référence

Liste des dates et heures
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

ErrObject, module (Visual Basic)

Le module **ErrObject** contient les propriétés et les procédures utilisées pour identifier et gérer les erreurs d'exécution à l'aide de l'objet **Err**.

Notes

Le module **ErrObject** prend en charge l'objet **Err**. Pour plus d'informations, consultez Err, objet (Visual Basic).

Membres

Clear	Description	Erl	GetException
HelpContext	HelpFile	LastDLLError	Number
Raise	Source		

Exemple

Cet exemple utilise les propriétés de l'objet **Err** pour construire une boîte de dialogue de message d'erreur. Notez ce qui se produit lorsque vous utilisez d'abord la méthode **Clear** : les valeurs par défaut de Visual Basic deviennent les propriétés de l'objet **Err** lorsque vous générez une erreur Visual Basic avec la méthode **Raise**.

```
Dim Msg As String
' If an error occurs, construct an error message.
On Error Resume Next   ' Defer error handling.
Err.Clear
Err.Raise(6)   ' Generate an "Overflow" error.
' Check for error, then show message.
If Err.Number <> 0 Then
    Msg = "Error # " & Str(Err.Number) & " was generated by " _
        & Err.Source & ControlChars.CrLf & Err.Description
    MsgBox(Msg, MsgBoxStyle.Information, "Error")
End If
```

Voir aussi

Référence

Err, objet (Visual Basic)
Liste des conversions
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

FileSystem, module (Visual Basic)

Le module **FileSystem** contient les procédures utilisées pour effectuer des opérations de fichier, répertoire ou dossier et de système.

La fonctionnalité **My** permet une meilleure productivité et de meilleures performances dans les opérations d'E/S de fichier que l'utilisation du module **FileSystem**. Pour plus d'informations, consultez My.Computer.FileSystem, objet.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque Runtime qui accèdent aux fichiers et aux dossiers.

Membres

ChDir	ChDrive	CurDir	Dir
EOF	FileAttr	FileClose	FileCopy
FileDateTime	FileGet	FileGetObject	FileLen
FileOpen	FilePut	FilePutObject	FileWidth
FreeFile	GetAttr	Entrée	InputString
Kill	LineInput	Loc	Verrouiller
LOF	MkDir	Print	PrintLine
Rename	Réinitialiser	RmDir	Seek
SetAttr	SPC	TAB	Déverrouiller
Write (écriture)	WriteLine		

Exemple

Cet exemple utilise la fonction **GetAttr** pour déterminer les attributs d'un fichier et un répertoire ou un dossier.

VB

```
Dim MyAttr As FileAttribute
' Assume file TESTFILE is normal and readonly.
MyAttr = GetAttr("C:\TESTFILE.txt")   ' Returns vbNormal.

' Test for normal.
If (MyAttr And FileAttribute.Normal) = FileAttribute.Normal Then
   MsgBox("This file is normal.")
End If

' Test for normal and readonly.
Dim normalReadonly As FileAttribute
normalReadonly = FileAttribute.Normal Or FileAttribute.ReadOnly
If (MyAttr And normalReadonly) = normalReadonly Then
   MsgBox("This file is normal and readonly.")
End If

' Assume MYDIR is a directory or folder.
MyAttr = GetAttr("C:\MYDIR")
If (MyAttr And FileAttribute.Directory) = FileAttribute.Directory Then
   MsgBox("MYDIR is a directory")
End If
```

Voir aussi

Référence

Liste des répertoires et des fichiers
Liste des entrées et sorties
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Financial, module (Visual Basic)

Le module **Financial** contient les procédures utilisées pour effectuer des opérations financières.

Notes

Ce module prend en charge les membres de la bibliothèque runtime Visual Basic qui exécutent des calculs financiers tels que l'amortissement, les valeurs actuelles et futures, les taux d'intérêt, les taux de retour et les paiements.

Membres

DDB	FV	IPmt	IRR
MIRR	NPer	NPV	Pmt
PPmt	PV	Rate	SLN
SYD			

Exemple

Cet exemple utilise la fonction **Rate** pour calculer le taux d'intérêt d'un emprunt, en fonction du nombre d'échéances (`TotPmts`), du montant des remboursements (`Payment`), de la valeur actuelle ou du capital de l'emprunt (`PVal`), de la valeur future de l'emprunt (`FVal`), d'un nombre indiquant si le paiement doit être effectué en début ou en fin de période (`PayType`), ainsi que de la valeur approximative du taux d'intérêt escompté (`Guess`).

VB

```
Sub TestRate()
    Dim PVal, Payment, TotPmts, APR As Double
    Dim PayType As DueDate

    ' Define percentage format.
    Dim Fmt As String = "##0.00"
    Dim Response As MsgBoxResult
    ' Usually 0 for a loan.
    Dim FVal As Double = 0
    ' Guess of 10 percent.
    Dim Guess As Double = 0.1
    PVal = CDbl(InputBox("How much did you borrow?"))
    Payment = CDbl(InputBox("What's your monthly payment?"))
    TotPmts = CDbl(InputBox("How many monthly payments do you have to make?"))
    Response = MsgBox("Do you make payments at the end of the month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    APR = (Rate(TotPmts, -Payment, PVal, FVal, PayType, Guess) * 12) * 100

    MsgBox("Your interest rate is " & Format(CInt(APR), Fmt) & " percent.")
End Sub
```

Voir aussi

Référence

Liste des mots clés financiers
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Globals, module (Visual Basic)

Le module **Globals** contient des fonctions de moteur de script.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime qui fournissent des informations sur le runtime actuellement utilisé.

Membres

ScriptEngine	ScriptEngineBuildVersion	ScriptEngineMajorVersion	ScriptEngineMinorVersion

Exemple

L'exemple ci-dessous utilise la propriété **ScriptEngineMajorVersion** et les propriétés associées pour retourner une chaîne décrivant les informations relatives au runtime utilisé :

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

Voir aussi

Référence

ScriptEngine, propriété
ScriptEngineBuildVersion, propriété
ScriptEngineMajorVersion, propriété
ScriptEngineMinorVersion, propriété
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Information, module (Visual Basic)

Le module **Information** contient les procédures utilisées pour retourner, tester ou vérifier les informations.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque Runtime qui fournissent des informations sur l'erreur, valident les données, déterminent les limites de tableau et récupèrent les informations sur la couleur et le type de données.

Membres

Err	IsArray	IsDate	IsDBNull
IsError	IsNothing	IsNumeric	IsReference
LBound	QBColor	RGB	SystemTypeName
TypeName	UBound	VarType	VbTypeName

Exemple

L'exemple suivant utilise la fonction **VbTypeName** pour retourner les noms de types de données de plusieurs variables.

```
Dim sysDateName As String = "System.DateTime"
Dim sysShortName As String = "Int16"
Dim sysBadName As String = "Nonsense"
Dim testVbName As String
testVbName = VbTypeName(sysDateName)
' Returns "Date".
testVbName = VbTypeName(sysShortName)
' Returns "Short".
testVbName = VbTypeName(sysBadName)
' Returns Nothing.
```

Voir aussi

Référence

Liste des informations et des interactions
Liste des types de données
Liste des déclarations et des constantes
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Interaction, module (Visual Basic)

Le module **Interaction** contient les procédures utilisées pour interagir avec les objets, les applications et les systèmes.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime qui exécutent d'autres programmes, appellent une méthode ou une propriété, émettent un signal sonore sur l'ordinateur, fournissent une chaîne de ligne de commande, manipulent des objets COM et contrôlent les boîtes de dialogue.

Membres

AppActivate	Beep	CallByName	Choose
Command	CreateObject	DeleteSetting	Environ
GetAllSettings	GetObject	GetSetting	IIf
InputBox	MsgBox	Partition	SaveSetting
Shell	Commutateur		

Exemple

L'exemple suivant utilise la fonction **Shell** pour exécuter une application spécifiée par l'utilisateur. Si Microsoft.VisualBasic.AppWinStyle.NormalFocus est spécifié en tant que second argument, l'application est activée et affichée sous sa taille normale.

```
Dim procID As Integer
' Run calculator.
procID = Shell("C:\Windows\system32\calc.exe", AppWinStyle.NormalFocus)
' The preceding path is for Windows XP.
' The Windows 2000 path is C:\WINNT\system32\calc.exe.
```

Voir aussi

Référence

Liste des informations et des interactions
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Strings, module (Visual Basic)

Le module **Strings** contient les procédures utilisées pour effectuer des opérations de chaînes.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime qui manipulent des chaînes.

Membres

Asc	AscW	Chr	ChrW
Filter	Format	FormatCurrency	FormatDateTime
FormatNumber	FormatPercent	GetChar	InStr
InStrRev	Join	LCase	Left
Len	LSet	LTrim	Mid
Replace	Right	RSet	RTrim
Espace	Split	StrComp	StrConv
StrDup	StrReverse	Trim	UCase

Exemple

L'exemple suivant montre comment fractionner une chaîne à ses espaces.

VB

```
Dim TestString As String = "Look at these!"
' Returns an array containing "Look", "at", and "these!".
Dim TestArray() As String = Split(TestString)
```

Voir aussi

Référence

Liste des manipulations de chaînes
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

VbMath, module (Visual Basic)

Le module **VbMath** contient les procédures utilisées pour effectuer des opérations mathématiques.

Notes

Ce module prend en charge les mots clés du langage Visual Basic et les membres de la bibliothèque runtime pour générer des nombres aléatoires.

Membres

Randomize	Rnd		

Exemple

Cet exemple utilise la fonction **Rnd** pour générer une valeur entière aléatoire comprise entre 1 et 6.

VB

```
' Initialize the random-number generator.
Randomize()
' Generate random value between 1 and 6.
Dim value As Integer = CInt(Int((6 * Rnd()) + 1))
```

Voir aussi

Référence

Liste des mots clés mathématiques
Fonctions mathématiques dérivées (Visual Basic)
Mots clés et membres par tâche
Mots clés du langage Visual Basic
Membres de la bibliothèque runtime Visual Basic
Comparaison des mots clés dans différents langages

Objets (Visual Basic)

Les rubriques de cette section documentent les objets runtime Visual Basic et contiennent des tables de leurs procédures, propriétés et événements membres.

Dans cette section

Collection, objet (Visual Basic)

Collection, membres de l'objet

Err, objet (Visual Basic)

Err, membres de l'objet

Rubriques connexes

Référence Visual Basic

Visual Basic

Collection, objet (Visual Basic)

Une **Collection** Visual Basic est un ensemble ordonné d'éléments pouvant être désigné comme une unité.

Notes

L'objet **Collection** Visual Basic offre un moyen pratique pour faire référence à un groupe d'éléments connexes comme un objet unique. Les *éléments* d'une collection doivent seulement être associés par leur existence dans la collection. Les éléments d'une collection ne doivent pas partager le même type de données.

Vous pouvez créer une collection de la même manière que vous créez d'autres objets, comme décrit dans l'exemple suivant.

```
Dim coll As New Microsoft.VisualBasic.Collection()
```

Une fois que vous avez créé une collection, vous pouvez :

- Ajouter un élément avec la méthode Add.
- Supprimez un élément avec la méthode Remove.
- Supprimez tous les éléments avec la méthode Clear.
- Connaître le nombre d'éléments contenus dans la collection avec la propriété Count.
- Vérifier la présence d'un élément spécifique avec la méthode Contains.
- Retourner un élément spécifique de la collection avec la propriété Item.
- Parcourir la collection entière avec For Each...Next, instruction (Visual Basic).

Remarque

Bien que l'objet **Collection** Visual Basic possède des fonctionnalités identiques à celles de l'objet **Collection** dans Visu al Basic 6.0, les deux ne peuvent pas interopérer dans un environnement COM.

Attention

Le parcours d'une **Collection** Visual Basic n'est pas une procédure thread-safe. Même si la collection est synchronisée, d'autres threads peuvent toujours la modifier, entraînant l'énumérateur à lever une exception. Pour garantir la sécurité des threads lors de l'énumération, verrouillez la collection ou interceptez les exceptions résultant des changements effe ctués par d'autres threads. Pour plus d'informations sur le verrouillage d'un élément de programmation, consultez SyncLock, instruction.

Exemple

L'exemple suivant crée les noms d'objets de la **Collection** et une boîte de dialogue pour que l'utilisateur ajoute des noms (`names`) d'objets à la collection. Il montre ensuite les noms de la collection, puis vide la collection sans effacer l'objet **Collection**.

Pour découvrir ce fonctionnement, sélectionnez la commande **Ajouter une classe** dans le menu **Projet** et déclarez une variable publique appelée `instanceName` au niveau du module de `nameClass` (type `Public instanceName`) pour contenir les noms de chaque instance. Conservez le nom par défaut (`nameClass`). Copiez et collez le code suivant dans la section **General** d'un autre module, puis faites-le démarrer par l'instruction `classNamer` dans une autre procédure. (Cet exemple fonctionne uniquement avec des applications hôtes qui prennent en charge des classes.)

VB

```
Public Class nameClass
    Public instanceName As String
End Class
Sub classNamer()
    ' Create a Visual Basic Collection object.
    Dim names As New Microsoft.VisualBasic.Collection()
    Dim key As Integer
    Dim msg As String
```

```
    Dim name As String
    Dim nameList As String = ""
    ' 1. Get names from the user to add to the collection.
    Do
        Dim inst As New nameClass()
        key += 1
        msg = "Please enter a name for this object." & vbCrLf _
            & "Press Cancel to see names in collection."
        name = InputBox(msg, "Name the Collection items")
        inst.instanceName = name
        ' If user entered a name, add it to the collection.
        If inst.instanceName <> "" Then
            names.Add(inst, CStr(key))
        End If
    Loop Until name = ""
    ' 2. Create and display a list of names from the collection.
    For Each oneInst As nameClass In names
        nameList &= oneInst.instanceName & vbCrLf
    Next oneInst
    MsgBox(nameList, , "Instance Names in names Collection")
    ' 3. Remove elements from the collection without disposing of the collection.
    For count As Integer = 1 To names.Count
        names.Remove(1)
        ' Since Visual Basic collections are reindexed automatically,
        ' remove the first member on each iteration.
    Next count
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Collection, membres de l'objet

GetEnumerator, méthode (objet Collection)

Collection, membres de l'objet

Un objet **Collection** Visual Basic expose les membres suivants :

Propriétés

Propriété	**Description**
Count, propriété	Lecture seule. Retourne un **Integer** qui contient le nombre d'éléments dans une collection Visual Basic.
Item, propriété	Lecture seule. Retourne un élément spécifique d'un objet **Collection** Visual Basic en fonction de sa position ou de sa clé.

Méthodes

Méthode	**Description**
Ajouter une méthode	Ajoute un élément à un objet **Collection** Visual Basic.
Clear, méthode	Supprime tous les éléments à partir d'un objet **Collection** Visual Basic.
Contains, méthode (objet Collection)	Retourne un **Boolean** indiquant si un objet **Collection** Visual Basic contient un élément avec une clé spécifique.
GetEnumerator, méthode	Retourne un objet énumérateur pour parcourir la collection.
Remove, méthode	Supprime un élément à partir d'un objet **Collection** Visual Basic.

Voir aussi

Référence

Collection, objet (Visual Basic)

Add, méthode (objet Collection)

Ajoute un élément à un objet **Collection**.

```
Public Sub Add( _
   ByVal Item As Object, _
   Optional ByVal Key As String, _
   Optional ByVal { Before | After } As Object = Nothing _
)
```

Paramètres

Item

Requis. Objet de tout type qui spécifie l'élément à ajouter à la collection.

Key

Facultatif. Expression **String** unique qui spécifie une chaîne clé qui peut être utilisée à la place d'un index positionnel pour accéder à ce nouvel élément dans la collection.

Before

Facultatif. Expression spécifiant une position relative dans la collection. L'élément à ajouter est placé dans la collection avant l'élément identifié par l'argument *Before*. Si *Before* est une expression numérique, il doit avoir pour valeur un nombre compris entre 1 et la valeur de la propriété Count, propriété (objet Collection) de la collection. Si *Before* est une expression **String**, il doit correspondre à la chaîne clé spécifiée au moment où l'élément auquel il est fait référence a été ajouté à la collection. Vous ne pouvez pas spécifier *Before* et *After* à la fois.

After

Facultatif. Expression spécifiant une position relative dans la collection. L'élément à ajouter est placé dans la collection après l'élément identifié par l'argument *After*. Si *After* est une expression numérique, il doit avoir pour valeur un nombre compris entre 1 et la valeur de la propriété **Count** de la collection. Si *After* est une expression **String**, il doit correspondre à la chaîne clé spécifiée au moment où l'élément auquel il est fait référence a été ajouté à la collection. Vous ne pouvez pas spécifier *Before* et *After* à la fois.

Exceptions/Codes d'erreur

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	• *Before* et *After* sont tous deux spécifiés. • L'argument *Before* ou *After* ne fait pas référence à un élément existant de la collection. • La *Key* spécifiée existe déjà.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

L'argument *Before* ou *After* doit faire référence à un élément existant de la collection ; sinon, une erreur se produit.

Une erreur se produit également si une valeur *Key* spécifiée correspond à la clé pour un élément existant de la collection.

Exemple

L'exemple suivant utilise la méthode **Add** pour ajouter des objets `child` (instances d'une classe appelée `child` contenant un `name` de propriété **Public**) à une collection appelée `family`. Pour voir comment cela fonctionne, créez un Form avec deux contrôles Button, puis affectez **Ajouter** et **Liste** à leurs propriétés Text. Ajoutez la définition de classe `child` et la déclaration `family` au code du formulaire. Modifiez les gestionnaires d'événements `_Click` pour les boutons **Ajouter** et **Liste** comme indiqué. Le bouton **Add** vous permet d'ajouter des enfants. Le bouton **Liste** permet d'afficher les noms de tous les enfants.

VB

```
Public Class child
    Public name As String
    Sub New(ByVal newName As String)
        name = newName
    End Sub
End Class
' Create a Collection object.
Private family As New Collection()
Private Sub addChild_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim newName As String
    newName = InputBox("Name of new family member: ")
    If newName <> "" Then
        family.Add(New child(newName), newName)
    End If
End Sub
Private Sub listChild_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click
    For Each aChild As child In family
        MsgBox(aChild.name)
    Next
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Collection, objet (Visual Basic)
Item, propriété (objet Collection)
Remove, méthode (objet Collection)

Clear, méthode (objet Collection)

Supprime tous les éléments d'un objet **Collection** Visual Basic.

```
Public Sub Clear()
```

Notes

La méthode **Clear** vide la collection et réinitialise sa propriété **Count** à 0.

Exemple
VB

```
Dim customers As New Microsoft.VisualBasic.Collection()
' Insert code that adds customers to collection.
customers.Clear()
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Collection, objet (Visual Basic)
Item, propriété (objet Collection)
Count, propriété (objet Collection)
Contains, méthode (objet Collection)
Add, méthode (objet Collection)
Remove, méthode (objet Collection)

Contains, méthode (objet Collection)

Retourne une valeur **Boolean** indiquant si un objet Visual Basic **Collection** contient un élément avec une clé spécifique.

```
Public Function Contains( _
    ByVal Key As String _
) As Boolean
```

Paramètres
Key

Obligatoire. Expression **String** qui spécifie la clé à rechercher dans les éléments de la collection.

Exceptions/Codes d'erreur

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	Le *Key* spécifié a la valeur **Nothing**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Contains retourne **True** si la collection contient un élément assorti d'une clé correspondant exactement à *Key*. Sinon, **Contains** retourne **False**.

Un **Collection** Visual Basic peut contenir certains éléments pourvus de clés et d'autres qui en sont dépourvus, selon que l'appel à Add, méthode (objet Collection) fournit un argument au paramètre facultatif *Key*.

Exemple
VB

```
Dim customers As New Microsoft.VisualBasic.Collection()
Dim accountNumber As String = "1234"
' Insert code that obtains new customer objects.
' Use the new customer's account number as the key.
customers.Add(newCustomer, accountNumber)
' The preceding statements can be repeated for several customers.
Dim searchNumber As String = "1234"
' Insert code to obtain an account number to search for.
If customers.Contains(searchNumber) Then
    MsgBox("The desired customer is in the collection.")
Else
    MsgBox("The desired customer is not in the collection.")
End If
```

Si vous envisagez de rechercher dans la collection des éléments à l'aide de leur clé, pensez à fournir l'argument *Key* chaque fois que vous appelez la méthode **Add**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Collection, objet (Visual Basic)
Item, propriété (objet Collection)
Count, propriété (objet Collection)

Add, méthode (objet Collection)
Remove, méthode (objet Collection)
Clear, méthode (objet Collection)

Count, propriété (objet Collection)

Retourne un **Integer** contenant le nombre d'éléments d'une collection. Lecture seule.

```
Public ReadOnly Property Count() As Integer
```

Notes

Utilisez la propriété **Count** pour déterminer le nombre d'éléments dans un objet **Collection**.

Exemple

Cet exemple illustre l'utilisation de la propriété **Count** pour afficher le nombre d'éléments dans une Collection, objet (Visual Basic) dans la variable `birthdays`.

VB

```
    Dim birthdays As New Collection()
    birthdays.Add(New DateTime(2001, 1, 12), "Bill")
    birthdays.Add(New DateTime(2001, 1, 13), "Joe")
    birthdays.Add(New DateTime(2001, 1, 14), "Mike")
    birthdays.Add(New DateTime(2001, 1, 15), "Pete")
...    MsgBox("Number of birthdays in collection: " & CStr(birthdays.Count))
```

L'objet **Collection** est de base 1, ce qui signifie que les valeurs d'index des éléments varient de 1 à la valeur de la propriété **Count**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Classe : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Collection, objet (Visual Basic)
Add, méthode (objet Collection)
Item, propriété (objet Collection)
Remove, méthode (objet Collection)

GetEnumerator, méthode (objet Collection)

Retourne une référence à un objet énumérateur utilisé pour parcourir un Collection, objet (Visual Basic).

```
Public Function GetEnumerator() As IEnumerator
```

Notes

For Each...Next, instruction (Visual Basic) appelle **GetEnumerator** pour obtenir un objet énumérateur pour prendre en charge l'itération sur les éléments d'une collection. Normalement, vous utilisez une boucle **For Each**...**Next** pour parcourir une collection ou un tableau sans devoir appeler **GetEnumerator** explicitement.

Si vous avez besoin de contrôle plus précis sur l'itération que celui fourni par les instructions **For Each**...**Next**, vous pouvez utiliser la méthode **GetEnumerator** pour exécuter un parcours personnalisé. Voici quelques cas pour lesquels vous serez susceptible de devoir utiliser cette méthode.

- Il est possible de retourner au début de la collection et de redémarrer l'itération avant qu'elle ne soit terminée.
- Vous est susceptible de vouloir ignorer un ou plusieurs éléments pour diverses raisons.
- Vous est susceptible de devoir modifier les éléments de la collection en plein milieu d'un parcours. Dans ce cas, vous devez obtenir un nouvel objet énumérateur, car le précédent n'est plus valable.

Exemple

L'exemple suivant montre comment utiliser **GetEnumerator** pour récupérer un élément de l'objet **Collection**.

VB

```
Dim customers As New Collection
' Insert code to add elements to the customers collection.
Dim custEnum As IEnumerator = customers.GetEnumerator()
custEnum.Reset()
Dim thisCustomer As Object
While custEnum.MoveNext()
    thisCustomer = custEnum.Current()
    ' Insert code to process this element of the collection.
End While
```

GetEnumerator construit et retourne un objet énumérateur qui implémente l'interface IEnumerator de l'espace de noms System.Collections. L'objet énumérateur expose la propriété Current et les méthodes MoveNext et Reset. Pour plus d'informations, consultez For Each...Next, instruction (Visual Basic).

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Collection, objet (Visual Basic)

Item, propriété (objet Collection)

Retourne un élément spécifique d'un objet **Collection** en fonction de sa position ou de sa clé. Lecture seule.

```
Default Public ReadOnly Property Item( _
    ByVal { Key As String | Index As Integer | Index As Object } _
) As Object
```

Paramètres

Key

Expression **String** unique qui spécifie une chaîne clé pouvant être utilisée à la place d'un index positionnel pour accéder à un élément de la collection. *Key* doit correspondre à l'argument *Key* spécifié lors de l'ajout de l'élément à la collection.

Index

(A) Expression numérique qui spécifie la position d'un élément de la collection. *Index* doit être un nombre compris entre 1 et la valeur du Count, propriété (objet Collection) de la collection. Ou (B) une expression **Object** qui spécifie la position ou chaîne clé d'un élément de la collection.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	• *Key* est non valide ou ne correspond à aucun membre existant de la collection. • *Index* ne peut pas être interprété comme caractère ou données numériques.
IndexOutOfRangeException	9	• Le *Key* a la valeur **Nothing**. • *Index* ne correspond à aucun membre existant de la collection.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Si *Index* est de type **Object**, la propriété **Item** essaie de le traiter comme un **String**, un **Char**, un tableau **Char**, ou une valeur entière. Si **Item** ne peut pas convertir *Index* en **String** ou **Integer**, il lève une exception **ArgumentException**.

La propriété **Item** est adoptée par défaut pour toutes les collections. Les lignes de code suivantes sont donc équivalentes.

```
MsgBox(CStr(customers.Item(1)))
MsgBox(CStr(customers(1)))
```

Exemple

Cet exemple utilise la propriété **Item** pour récupérer une référence à un objet de collection. Elle crée `birthdays` comme un objet **Collection** puis récupère l'objet représentant l'anniversaire de Bill, à l'aide de la clé `"Bill"` comme argument *Index*.

VB

```
    Dim birthdays As New Collection()
    birthdays.Add(New DateTime(2001, 1, 12), "Bill")
    birthdays.Add(New DateTime(2001, 1, 13), "Joe")
    birthdays.Add(New DateTime(2001, 1, 14), "Mike")
    birthdays.Add(New DateTime(2001, 1, 15), "Pete")
...    Dim aBirthday As DateTime
```

```
aBirthday = birthdays.Item("Bill")
MsgBox(CStr(aBirthday))
aBirthday = birthdays("Bill")
MsgBox(CStr(aBirthday))
```

Notez que le premier appel spécifie explicitement la propriété **Item**, contrairement au second. Les deux appels aboutissent, car la propriété **Item** est la propriété par défaut pour un objet **Collection**.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Collection, objet (Visual Basic)
Add, méthode (objet Collection)
Count, propriété (objet Collection)
Remove, méthode (objet Collection)

Remove, méthode (objet Collection)

Supprime un élément d'un objet **Collection**.

```
Public Overloads Sub Remove(_
    ByVal { Key As String | Index As Integer } _
)
```

Paramètres

Key

Expression **String** unique qui spécifie une chaîne clé pouvant être utilisée à la place d'un index positionnel pour accéder à un élément de la collection. *Key* doit correspondre à l'argument *Key* spécifié lors de l'ajout de l'élément à la collection.

Index

Expression numérique qui spécifie la position d'un élément de la collection. *Index* doit être un nombre compris entre 1 et la valeur du Count, propriété (objet Collection) de la collection.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Key* n'est pas valide ou n'est pas spécifié.
IndexOutOfRangeException	9	*Index* ne correspond à aucun membre existant de la collection.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Lorsque **Remove** supprime un élément d'une collection, il décrémente la collection Count, propriété (objet Collection) de un. Il décrémente également la valeur *Index* de chaque élément qui suivait précédemment l'élément supprimé dans la collection.

Si un élément a été ajouté à la collection sans *Key*, vous devez utiliser son *Index* pour le supprimer.

Exemple

Cet exemple montre comment utiliser la méthode **Remove** pour supprimer des objets d'un Collection, objet (Visual Basic) de la variable `birthdays`.

VB

```
    Dim birthdays As New Collection()
    birthdays.Add(New DateTime(2001, 1, 12), "Bill")
    birthdays.Add(New DateTime(2001, 1, 13), "Joe")
    birthdays.Add(New DateTime(2001, 1, 14), "Mike")
    birthdays.Add(New DateTime(2001, 1, 15), "Pete")
...    birthdays.Remove(1)
    birthdays.Remove("Mike")
```

À la suite des quatre appels à la méthode **Add**, la propriété **Count** contient 4, le `"Bill"` d'élément a la valeur d'index 1, et le `"Pete"` d'élément a la valeur d'index 4.

À la suite du premier appel à **Remove**, **Count** est égal à 3, le `"Bill"` d'élément est supprimé, et le `"Pete"` d'élément a la valeur d'index 3.

À la suite du second appel à **Remove**, **Count** est égal à 2, le `"Mike"` d'élément est supprimé, et le `"Pete"` d'élément a la valeur d'index 2.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Collection

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Référence
Collection, objet (Visual Basic)
Add, méthode (objet Collection)

Err, objet (Visual Basic)

Contient des informations sur les erreurs d'exécution.

Notes

Les propriétés de l'objet **Err** sont définies par le générateur d'une erreur (Visual Basic, un objet ou le programmeur).

Lorsqu'une erreur d'exécution se produit, les propriétés de l'objet **Err** sont complétées par des informations qui identifient l'erreur de manière unique et que vous pouvez utiliser pour gérer l'erreur. Pour générer une erreur d'exécution dans votre code, utilisez la méthode **Raise**.

La valeur zéro ou des chaînes de longueur nulle ("") sont réaffectées aux propriétés de l'objet **Err** après une instruction **Exit Sub**, **Exit Function**, **Exit Property** ou **Resume Next** dans une routine de gestion des erreurs. L'utilisation d'un formulaire de l'instruction **Resume** en dehors d'une routine de gestion des erreurs ne réinitialisera pas les propriétés de l'objet **Err**. Vous pouvez utiliser la méthode **Clear** pour réinitialiser explicitement **Err**.

Utilisez la méthode **Raise** plutôt que l'instruction **Error** pour générer des erreurs d'exécution pour les erreurs système et les modules de classe. Votre décision d'utiliser ou non la méthode **Raise** dans un autre code dépend de la richesse des informations que vous souhaitez retourner.

L'objet **Err** est un objet intrinsèque avec une portée globale. Vous ne devez donc pas créer d'instance de cet objet dans votre code.

Exemple

Cet exemple utilise les propriétés de l'objet **Err** pour construire une boîte de dialogue de message d'erreur. Notez que si vous utilisez d'abord la méthode **Clear**, les valeurs par défaut de Visual Basic deviennent les propriétés de l'objet **Err** lorsque vous générez une erreur Visual Basic avec la méthode **Raise**.

VB

```
Dim Msg As String
' If an error occurs, construct an error message.
On Error Resume Next   ' Defer error handling.
Err.Clear()
Err.Raise(6)   ' Generate an "Overflow" error.
' Check for error, then show message.
If Err.Number <> 0 Then
  Msg = "Error # " & Str(Err.Number) & " was generated by " _
      & Err.Source & ControlChars.CrLf & Err.Description
  MsgBox(Msg, MsgBoxStyle.Information, "Error")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic

Comment : extraire des informations à partir d'un objet Error

Référence

Err, membres de l'objet

Error, instruction

On Error, instruction (Visual Basic)

Exit, instruction (Visual Basic)

Resume, instruction

Err, membres de l'objet

L'objet **Err** contient des propriétés dont les valeurs sont déterminées par l'erreur qui vient de se produire. La propriété **Number** contient la cause de l'erreur et la propriété **Description** contient un message texte qui décrit l'erreur en détail. **Helpfile** et **HelpContext** vous permettent d'afficher un fichier d'aide connexe lorsqu'un utilisateur clique sur le bouton **Help** ou appuie sur la touche F1. **LastDLLError** affiche la dernière DLL appelée et indique si l'appel a réussi ou échoué. **Source** spécifie une expression de chaîne représentant l'objet ou l'application ayant généré l'erreur.

Propriétés

Description, propriété	Lecture/écriture. Retourne ou définit une chaîne de description associée à une erreur.
Erl, propriété	Lecture seule. Retourne un entier indiquant le numéro de ligne de la dernière instruction exécutée.
HelpContext, propriété	Lecture/écriture. Retourne ou définit une valeur de type **Integer** contenant l'ID de contexte d'une rubrique de fichier d'aide.
HelpFile, propriété	Lecture/écriture. Retourne ou définit une expression **String** contenant le chemin d'accès complet vers un fichier d'aide.
LastDLLError, propriété	Lecture seule. Retourne le code d'erreur système généré par un appel à une DLL (bibliothèque de liens dynamiques).
Number, propriété	Lecture/écriture. Retourne ou spécifie une valeur numérique spécifiant une erreur.
Source, propriété	Lecture/écriture. Retourne ou définit une expression **String** spécifiant le nom de l'objet ou de l'application qui est à l'origine de l'erreur.

Méthodes

Clear, méthode	Efface tous les paramètres de propriété de l'objet **Err**.
Raise, méthode	Génère une erreur d'exécution ; peut être utilisée à la place de l'instruction **Error**.

Voir aussi

Tâches

Comment : extraire des informations à partir d'un objet Error

Référence

Err, objet (Visual Basic)

Clear, méthode (objet Err)

Efface tous les paramètres de propriété de l'objet **Err**.

```
object.Clear
```

Paramètres
object

Toujours l'objet **Err**.

Notes

Utilisez **Clear** pour effacer explicitement l'objet **Err** après le traitement d'une erreur, par exemple lorsque vous utilisez le traitement différé d'une erreur avec **On Error Resume Next**. La méthode **Clear** est appelée automatiquement chaque fois que l'une des instructions suivantes est exécutée :

- Tout type d'instruction **Resume**.
- **Exit Sub**, **Exit Function** ou **Exit Property**
- Toute instruction **On Error**.
- Toute instruction **Try...Catch...Finally**.

> **Remarque**
>
> La construction **On Error Resume Next** peut être préférable à **On Error GoTo** lors de la gestion des erreurs générées au cours de l'accès à d'autres objets. La vérification de **Err** après chaque interaction avec un objet lève toute ambiguïté concernant l'objet auquel le code a accédé : vous pouvez connaître avec certitude l'objet qui a placé le code d'erreur dans **Err.Number** ainsi que celui qui a généré l'erreur à l'origine (l'objet spécifié dans **Err.Source**).

Exemple

Cet exemple utilise la méthode **Clear** de l'objet **Err** pour réinitialiser les propriétés numériques de l'objet **Err** et affecter à ses propriétés de type chaîne des chaînes nulles. Sans l'appel à **Clear**, le deuxième appel à **MsgBox** afficherait le même message d'erreur.

VB

```
Sub ClearErr()
  ' Produce overflow error
  On Error Resume Next
  Dim zero As Integer = 0
  Dim result As Integer = 8 / zero
  MsgBox(Err.Description)
  Err.Clear()
  MsgBox(Err.Description)
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi
Tâches
Comment : extraire des informations à partir d'un objet Error
Référence
Err, objet (Visual Basic)

Description, propriété (objet Err)
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
On Error, instruction (Visual Basic)
Raise, méthode (objet Err)
Source, propriété (objet Err)

Description, propriété (objet Err)

Retourne ou définit une expression **String** contenant une chaîne descriptive associée à une erreur. Lecture/écriture.

```
Public Property Description() As String
```

Notes

Le paramètre de propriété **Description** est une courte description de l'erreur. Utilisez cette propriété pour avertir l'utilisateur d'une erreur que vous ne pouvez pas ou ne voulez pas gérer. Lors de la génération d'une erreur définie par l'utilisateur, assignez une courte description de votre erreur à la propriété **Description**. Si la propriété **Description** reste vide et que la valeur de la propriété Number correspond à une erreur d'exécution Visual Basic, la chaîne retournée par la fonction ErrorToString est définie dans la propriété **Description** lorsque l'erreur est générée.

Exemple

Cet exemple assigne un message défini par l'utilisateur à la propriété **Description** de l'objet **Err**.

VB

```
On Error Resume Next
Err.Raise(60000)
Err.Description = "Your Widget needs a new Frob!"
MsgBox(Err.Description)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
ErrorToString, fonction
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
Source, propriété (objet Err)

Erl, propriété (objet Err)

Retourne un entier indiquant le numéro de ligne de la dernière instruction exécutée. Lecture seule.

```
Public ReadOnly Property Erl() As Integer
```

Notes

Si Visual Basic ne rencontre aucun numéro de ligne, il retourne la valeur 0.

Exemple

Cet exemple utilise la propriété **Erl** pour indiquer le numéro de ligne.

VB

```
10:     On Error Resume Next
20:     Err.Raise(60000)
' Returns 20.
30:     MsgBox(Erl())
```

Notes du développeur sur Smart Device

Cette propriété n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
ErrorToString, fonction

HelpContext, propriété (objet Err)

Retourne ou définit une valeur de type **Integer** contenant l'ID de contexte d'une rubrique de fichier d'aide. Lecture/écriture.

```
Public Property HelpContext() As Integer
```

Notes

La propriété **HelpContext** permet d'afficher l'aide contextuelle d'une application. Si un fichier d'aide est spécifié dans **HelpFile**, la propriété **HelpContext** permet d'afficher automatiquement le fichier d'aide identifié. Si les propriétés **HelpFile** et **HelpContext** sont vides, la valeur de la propriété Number est vérifiée. Si la valeur de la propriété **Number** correspond à une valeur d'erreur d'exécution Visual Basic, c'est l'ID de contexte d'aide Visual Basic de cette erreur qui est utilisé. Si la valeur de la propriété **Number** ne correspond pas à une erreur Visual Basic, l'écran du sommaire du fichier d'aide de Visual Basic s'affiche.

Remarque

Vous devez écrire des routines dans votre application pour gérer les erreurs courantes. Pour la programmation avec un objet , vous pouvez utiliser le fichier d'aide de l'objet pour améliorer la qualité de votre gestion des erreurs ou bien pour afficher u n message significatif à l'attention de l'utilisateur si l'erreur n'est pas récupérable.

Exemple

Cet exemple utilise la propriété **HelpContext** de l'objet **Err** pour afficher la rubrique d'aide Visual Basic correspondant à l'erreur `Overflow`.

VB

```
Dim Msg As String
Err.Clear()
On Error Resume Next   ' Suppress errors for demonstration purposes.
Err.Raise(6)   ' Generate "Overflow" error.
If Err.Number <> 0 Then
  Msg = "Press F1 or HELP to see " & Err.HelpFile & " topic for" & _
  " the following HelpContext: " & Err.HelpContext
  MsgBox(Msg, , "Error:")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
Description, propriété (objet Err)
ErrorToString, fonction
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
Source, propriété (objet Err)

HelpFile, propriété (objet Err)

Retourne ou définit une expression **String** contenant le chemin d'accès complet vers un fichier d'aide. Lecture/écriture.

```
Public Property HelpFile() As String
```

Notes

Si un fichier d'aide est spécifié par la propriété **HelpFile**, il est automatiquement appelé lorsque l'utilisateur appuie sur la touche Aide (ou sur la touche F1) dans la boîte de dialogue de message d'erreur. Si la propriété **HelpContext** contient un ID de contexte valide pour le fichier spécifié, cette rubrique est automatiquement affichée. Si aucune propriété **HelpFile** n'est spécifiée, le fichier d'aide de Visual Basic s'affiche.

Remarque

Vous devez écrire des routines dans votre application pour gérer les erreurs courantes. Pour la programmation avec un objet , vous pouvez utiliser le fichier d'aide de l'objet pour améliorer la qualité de votre gestion des erreurs ou bien pour afficher u n message significatif à l'attention de l'utilisateur si l'erreur n'est pas récupérable.

Exemple

Cet exemple utilise la propriété **HelpFile** de l'objet **Err** pour démarrer le système d'aide. Par défaut, la propriété **HelpFile** contient le nom du fichier d'aide de Visual Basic.

VB

```
Dim Msg As String
Err.Clear()
On Error Resume Next   ' Suppress errors for demonstration purposes.
Err.Raise(6)   ' Generate "Overflow" error.
If Err.Number <> 0 Then
  Msg = "Press F1 or HELP to see " & Err.HelpFile & " topic for" & _
  " the following HelpContext: " & Err.HelpContext
  MsgBox(Msg, , "Error:")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
Description, propriété (objet Err)
HelpContext, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
Source, propriété (objet Err)

LastDllError, propriété (objet Err)

Retourne le code d'erreur système généré par un appel à une DLL (bibliothèque de liens dynamiques). Lecture seule.

```
ReadOnly Property LastDllError() As Integer
```

Notes

La propriété **LastDllError** s'applique uniquement aux appels de DLL effectués à partir de code Visual Basic. Lorsque ce type d'appel est effectué, la fonction appelée retourne généralement un code indiquant le succès ou l'échec et une valeur est assignée à la propriété **LastDllError**. Consultez la documentation des fonctions de la DLL afin de déterminer les valeurs de retour indiquant le succès ou l'échec. À chaque fois que le code d'échec est retourné, l'application Visual Basic doit vérifier immédiatement la propriété **LastDllError**. Aucune exception n'est déclenchée lorsque la propriété **LastDllError** est définie.

Remarque

La propriété **LastDllError** nécessite une autorisation de code non managée, qui peut affecter son exécution dans les situation s d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

L'exemple suivant montre comment vous pouvez utiliser la propriété **LastDllError** après avoir appelé une fonction dans l'API Windows. La procédure **PrintWindowCoordinates** accepte un handle dans une fenêtre et appelle la fonction **GetWindowRect**. **GetWindowRect** remplit la structure de données RECT avec les longueurs des côtés du rectangle qui compose la fenêtre. Si vous passez un handle qui n'est pas valide, une erreur se produit, laissant apparaître le numéro d'erreur via la propriété **LastDllError**.

VB

```
  Declare Function GetWindowRect Lib "user32" _
  (ByVal hwnd As Integer, ByRef lpRect As RECT) As Integer
...
  Public Structure RECT
      Public Left As Integer
      Public Top As Integer
      Public Right As Integer
      Public Bottom As Integer
  End Structure
...
  Const ERROR_INVALID_WINDOW_HANDLE As Long = 1400
  Const ERROR_INVALID_WINDOW_HANDLE_DESCR As String = _
  "Invalid window handle."
```

VB

```
Private Sub PrintWindowCoordinates(ByVal hwnd As Integer)
' Prints left, right, top, and bottom positions
' of a window in pixels.

  Dim rectWindow As RECT

  ' Pass in window handle and empty the data structure.
  ' If function returns 0, an error occurred.
  If GetWindowRect(hwnd, rectWindow) = 0 Then
      ' Check LastDllError and display a dialog box if the error
      ' occurred because an invalid handle was passed.
      If Err.LastDllError = ERROR_INVALID_WINDOW_HANDLE Then
          MsgBox(ERROR_INVALID_WINDOW_HANDLE_DESCR, Title:="Error!")
      End If
  Else
      Debug.Print(rectWindow.Bottom)
      Debug.Print(rectWindow.Left)
```

```
        Debug.Print(rectWindow.Right)
        Debug.Print(rectWindow.Top)
    End If
End Sub
```

Notes du développeur sur Smart Device

Cette propriété retourne toujours zéro.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
Declare, instruction
Description, propriété (objet Err)
ErrorToString, fonction
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
Number, propriété (objet Err)
Source, propriété (objet Err)

Number, propriété (objet Err)

Retourne ou spécifie une valeur numérique spécifiant une erreur. Lecture/écriture.

```
Public Property Number() As Integer
```

Notes

Pour retourner à partir d'un objet une erreur définie par l'utilisateur, définissez **Err.Number** en ajoutant à la constante **VbObjectError** le nombre que vous avez choisi comme code d'erreur. Par exemple, utilisez le code suivant pour retourner le nombre 1 051 comme code d'erreur :

VB

```
Err.Raise(Number:=vbObjectError + 1051, Source:="SomeClass")
```

Exemple

Cet exemple illustre une utilisation classique de la propriété **Number** dans une routine de gestion des erreurs.

VB

```
' Typical use of Number property.
Sub test()
  On Error GoTo out

  Dim x, y As Integer
  x = 1 / y   ' Create division by zero error.
  Exit Sub
out:
  MsgBox(Err.Number)
  MsgBox(Err.Description)
  ' Check for division by zero error.
  If Err.Number = 11 Then
      y = y + 1
  End If
  Resume Next
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
Description, propriété (objet Err)
ErrorToString, fonction
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Source, propriété (objet Err)

Raise, méthode (objet Err)

Génère une erreur d'exécution ; peut être utilisée à la place de l'instruction **Error**.

```
Public Sub Raise( _
   ByVal Number As Integer, _
   Optional ByVal Source As Object = Nothing, _
   Optional ByVal Description As Object = Nothing, _
   Optional ByVal HelpFile As Object = Nothing, _
   Optional ByVal HelpContext As Object = Nothing _
)
```

Paramètres

Number

Requis. Entier **Long** qui identifie la nature de l'erreur. Les codes d'erreur Visual Basic doivent être compris entre 0 et 65 535 : les codes compris entre 0 et 512 sont réservés aux erreurs système ; ceux compris entre 513 et 65 535 peuvent être utilisés pour les erreurs définies par l'utilisateur. Lorsque vous définissez la propriété **Number** pour votre propre code d'erreur dans un module de classe, ajoutez le numéro de votre code d'erreur à la constante **vbObjectError**. Par exemple, pour générer le numéro d'erreur 513, assignez `vbObjectError + 513` à la propriété **Number**.

Source

Facultatif. Expression **String** attribuant un nom à l'objet ou à l'application ayant généré l'erreur. Lorsque vous définissez cette propriété pour un objet, utilisez le formulaire *project.class*. Si *Source* n'est pas spécifié, l'ID de processus du projet Visual Basic actuel est utilisé.

Description

Facultatif. Expression **String** décrivant l'erreur. Si aucune description n'est spécifiée, la valeur contenue dans la propriété **Number** est examinée. Si la description indiquée peut être mappée à un code d'erreur d'exécution Visual Basic, la chaîne retournée par la fonction **Error** est utilisée comme la propriété **Description**. En l'absence d'erreur Visual Basic correspondant à la propriété **Number**, le message "Erreur définie par l'application ou par l'objet" est utilisé.

HelpFile

Facultatif. Chemin d'accès complet au fichier contenant l'aide relative à cette erreur. Si aucun fichier d'aide n'est spécifié, Visual Basic utilise le lecteur, le chemin d'accès et le nom de fichier complets du fichier d'aide Visual Basic.

HelpContext

Facultatif. ID de contexte identifiant une rubrique contenue dans *HelpFile* qui fournit l'aide correspondant à l'erreur. S'il est omis, l'ID de contexte qui est utilisé est celui du fichier d'aide Visual Basic pour l'erreur correspondant à la propriété **Number**, s'il existe.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Number* est supérieur à 65 535.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Tous les arguments **Raise** à l'exception de *Number* sont facultatifs. Si vous omettez les arguments facultatifs et si les paramètres de propriété de l'objet **Err** contiennent des valeurs qui n'ont pas été effacées, celles-ci servent de valeurs pour votre erreur.

Comme l'objet **Err** donne des informations plus riches que lorsque vous générez des erreurs avec l'instruction **Error**, **Raise** est utile pour générer des erreurs lors de l'écriture de modules de classe. Par exemple, avec la méthode **Raise**, la source ayant généré l'erreur peut être spécifiée dans la propriété **Source**, l'aide en ligne correspondant à l'erreur peut être référencée, etc.

Exemple

Cet exemple utilise la méthode **Raise** de l'objet **Err** pour générer une erreur à l'intérieur d'une fonction écrite en Visual Basic. La fonction appelante peut intercepter l'erreur et la signaler à l'utilisateur à l'aide d'un message.

VB

```
Const WidthError As Integer = 1
Const WidthHelp As Object = 101

Sub TestWidth(ByVal width As Integer)
  If width > 1000 Then
      Err.Raise(vbObjectError + 512 + WidthError, "TestWidth", _
      "Width must be less than 1000.", "HelpFile.hlp", WidthHelp)
  End If
End Sub

Sub CallingProcedure()
  Try
    TestWidth(2000)
  Catch ex As Exception
    MsgBox(ex.Message)
  End Try
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
Clear, méthode (objet Err)
Description, propriété (objet Err)
Error, instruction
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
On Error, instruction (Visual Basic)
Source, propriété (objet Err)
ArgumentException Class

Source, propriété (objet Err)

Retourne ou définit une expression **String** spécifiant le nom de l'objet ou de l'application qui est à l'origine de l'erreur. Lecture/écriture.

```
Public Property Source() As String
```

Notes

La propriété **Source** spécifie une expression **String** représentant l'objet qui a généré l'erreur ; l'expression est généralement le nom de la classe de l'objet ou l'ID de processus. Utilisez la propriété **Source** pour fournir des informations lorsque votre code est incapable de prendre en charge une erreur générée dans un objet utilisé. Par exemple, si vous accédez à Microsoft Excel et que celui-ci génère une erreur `Division by zero`, il affecte à **Err.Number** le code de cette erreur et à la propriété **Source** la chaîne "Excel.Application".

Lorsque l'application génère une erreur à partir du code, **Source** est l'ID programmatique de votre application. Dans une classe, **Source** doit contenir un nom de la forme *project.class*. Lorsqu'une erreur inattendue se produit dans votre code, la propriété **Source** est remplie automatiquement. Pour les erreurs produites dans un module, **Source** contient le nom du projet.

Exemple

Cet exemple illustre l'utilisation de la propriété **Source** dans une routine classique de gestion des erreurs. Lorsqu'une erreur se produit à partir de `Class1`, la chaîne `"Class1"` est assignée à la propriété **Source** de l'objet **Err**. Cette chaîne est ensuite affichée dans un message d'information indiquant la source et le numéro de l'erreur.

VB

```
Public Class Class1
  Public Sub MySub()
      On Error Resume Next
      Err.Raise(60000, "Class1")
      MsgBox(Err.Source & " caused an error of type " & Err.Number)
  End Sub
End Class
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : ErrObject

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Err, objet (Visual Basic)
Description, propriété (objet Err)
ErrorToString, fonction
GetObject, fonction (Visual Basic)
HelpContext, propriété (objet Err)
HelpContext, propriété (objet Err)
HelpFile, propriété (objet Err)
LastDllError, propriété (objet Err)
Number, propriété (objet Err)
On Error, instruction (Visual Basic)

My.Application, objet

Fournit des propriétés, des méthodes et des événements en rapport avec l'application en cours.

Notes

Les propriétés exposées par l'objet **My.Application** retournent les données associées uniquement à l'application ou la DLL actuelle. Aucune information au niveau du système ne peut être modifiée avec **My.Application**.

Certains membres sont uniquement disponibles pour les applications console ou Windows Forms.

Exemple

Cet exemple utilise la propriété **My.Application.CommandLineArgs** pour examiner les arguments de ligne de commande de l'application. Si un argument détecté démarre avec `/input=`, le reste de cet argument est affiché.

VB

```
Private Sub ParseCommandLineArgs()
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In My.Application.CommandLineArgs
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next

    If inputName = "" Then
        MsgBox("No input name")
    Else
        MsgBox("Input name: " & inputName)
    End If
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase (la classe de base ConsoleApplicationBase fournit les membres disponibles dans les applications console, et sa classe de base ApplicationBase fournit les membres disponibles dans tous les projets)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Application, membres de l'objet
My.Application.ApplicationContext, propriété
My.Application.Info, objet
My.Application.Log, objet
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase
Microsoft.VisualBasic.ApplicationServices.ApplicationBase

My.Application, membres de l'objet

L'pbjet My.Application fournit des propriétés, des méthodes et des événements en rapport avec l'application en cours.

Propriétés

Propriété	Description
ApplicationContext	Obtient l'objet ApplicationContext pour le thread actuel d'une application Windows Forms . Cette propriété n'est disponible que pour les applications Windows Forms.
CommandLineArgs	Obtient une collection qui contient les arguments de ligne de commande sous forme de c haînes pour l'application actuelle.
Culture	Obtient la culture utilisée par le thread actuel pour la manipulation et la mise en forme de chaînes.
Déploiement	Obtient l'objet de déploiement ClickOnce de l'application actuelle, qui prend en charge la mise à jour du déploiement actuel par programme et du téléchargement de fichiers à la d emande.
Info	Retourne l'objet My.Application.Info, qui fournit des propriétés pour accéder aux informati ons sur l'assembly de l'application, par exemple le numéro de version, la description, etc.
IsNetworkDeployed	Obtient **Boolean** qui indique si l'application a été déployée dans un réseau.
Log	Retourne l'objet My.Application.Log, qui fournit une propriété et des méthodes pour écrire des informations sur les événements et les exceptions dans les écouteurs de journalisation des événements de l'application.
MinimumSplashScreenDisplayTime	Obtient ou définit la durée minimale, en millisecondes, pendant laquelle l'écran de démarr age s'affiche.
OpenForms	Obtient une collection des formulaires ouverts de l'application. Cette propriété n'est disponible que pour les applications Windows Forms.
SaveMySettingsOnExit	Détermine si l'application enregistre les paramètres utilisateur en quittant. Cette propriété n'est disponible que pour les applications console et Windows Forms.
SplashScreen	Obtient ou définit l'écran de démarrage pour cette application. Cette propriété n'est disponible que pour les applications Windows Forms.
UICulture	Obtient la culture utilisée par le thread actuel pour récupérer des ressources propres à la c ulture.

Méthodes

Méthode	Description
ChangeCulture	Modifie la culture utilisée par le thread actuel pour la manipulation et la mise en forme de chaînes.
ChangeUICulture	Modifie la culture utilisée par le thread actuel pour récupérer des ressources propres à la culture.
DoEvents	Traite tous les messages Windows qui sont actuellement contenus dans la file d'attente de messages. Cette méthode n'est disponible que pour les applications Windows Forms.

GetEnvironmentVariable	Retourne la valeur de la variable d'environnement spécifiée.
Run	Configure et démarre le modèle d'application Visual Basic. Cette méthode n'est disponible que pour les applications Windows Forms.

Événements

Événement	**Description**
NetworkAvailabilityChanged	Se produit en cas de modification de la disponibilité du réseau. Cet événement n'est disponible que pour les applications Windows Forms.
Arrêt	Se produit à l'arrêt de l'application. Cet événement n'est disponible que pour les applications Windows Forms.
Startup	Se produit au démarrage de l'application. Cet événement n'est disponible que pour les applications Windows Forms.
StartupNextInstance	Se produit lors d'une tentative de démarrage d'une application à instance unique et lorsque l'application est déjà active. Cet événement n'est disponible que pour les applications Windows Forms.
UnhandledException	Se produit lorsque l'application rencontre une exception non gérée. Cet événement n'est disponible que pour les applications Windows Forms.

Voir aussi

Référence

My.Application, objet

My.Application.ApplicationContext, propriété

Obtient l'objet ApplicationContext pour le thread actuel d'une application Windows Forms.

```
' Usage
Dim value As System.Windows.Forms.ApplicationContext = My.Application.ApplicationContext
' Declaration
Public ReadOnly Property ApplicationContext As System.Windows.Forms.ApplicationContext
```

Valeur de retour

Cette propriété retourne l'objet **ApplicationContext** du thread actuel. Cet objet contient des informations contextuelles concernant le thread.

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet
System.Windows.Forms.ApplicationContext
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.ApplicationContext

My.Application.ChangeCulture, méthode

Modifie la culture utilisée par le thread actuel pour la manipulation et la mise en forme de chaînes.

```
' Usage
My.Application.ChangeCulture(cultureName)
' Declaration
Public Sub ChangeCulture( _
   ByVal cultureName As String _
)
```

Paramètres

cultureName

String. Nom de la culture sous forme de chaîne. Pour obtenir la liste des noms possibles, consultez CultureInfo.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- L'argument *cultureName* a la valeur **Nothing** (ArgumentNullException).
- L'argument *cultureName* n'est pas un nom de culture valide (ArgumentException).

Notes

La méthode **My.Application.ChangeCulture** modifie la propriété CurrentCulture du thread actuel. **CurrentCulture** détermine les formats par défaut pour les dates, les heures, la monnaie, les chiffres, l'ordre de tri du texte, les comparaisons de chaînes et la casse pour tous les calculs sur ce thread.

Pour récupérer la culture actuelle, vous pouvez utiliser la propriété My.Application.Culture ou **CurrentCulture**.

Le paramètre **CurrentCulture** est différent d'un paramètre de langue. Elle contient uniquement des données liées aux paramètres standard en vigueur dans une région géographique. Par conséquent, seules une culture spécifique ou InvariantCulture peuvent être affectées à la propriété **CurrentCulture**.

Utilisez la méthode **My.Application.ChangeUICulture** pour modifier la culture utilisée par le thread actuel pour récupérer des ressources propres à la culture.

Remarque de sécurité

La méthode **My.Application.ChangeCulture** requiert SecurityPermission avec le jeu ControlThread. L'état de sécurité associé aux threads rend la manipulation de threads dangereuse. Par conséquent, cette autorisation ne doit être accordée qu'au code fiable et uniquement en cas de nécessité. Vous ne pouvez pas changer la culture d'un thread dans du code de niveau de confiance partiel.

Exemple

Cet exemple montre comment la modification de la culture change la représentation sous forme de chaîne des dates.

VB

```
Private Sub TestChangeCulture()
    ' Store the current culture.
    Dim currentculture As String = My.Application.Culture.Name
    MsgBox("Current culture is " & currentculture)

    Dim jan1 As New Date(2005, 1, 1, 15, 15, 15)

    My.Application.ChangeCulture("en-US")
    MsgBox("Date represented in en-US culture: " & jan1)
    ' 1/1/2005 3:15:15 PM

    My.Application.ChangeCulture("")
    MsgBox("Date represented in invariant culture" & jan1)
```

```
        ' 01/01/2005 15:15:15

        ' Restore the culture.
        My.Application.ChangeCulture(currentculture)
    End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase, ApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : **ControlThread**.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet
My.Application.Culture, propriété
My.Application.ChangeUICulture, méthode
Microsoft.VisualBasic.ApplicationServices.ApplicationBase.ChangeCulture(System.String)
CurrentCulture

My.Application.ChangeUICulture, méthode

Modifie la culture que le thread actuel utilise pour récupérer des ressources spécifiques à la culture.

```
' Usage
My.Application.ChangeUICulture(cultureName)
' Declaration
Public Sub ChangeUICulture( _
   ByVal cultureName As String _
)
```

Paramètres

cultureName

String. Nom de la culture sous forme de chaîne. Pour obtenir la liste des noms possibles, consultez CultureInfo.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- L'argument *cultureName* a la valeur **Nothing** (ArgumentNullException).
- L'argument *cultureName* n'est pas un nom de culture valide (ArgumentException).

Notes

La méthode **My.Application.ChangeUICulture** modifie la propriété CurrentUICulture du thread actuel. La propriété **CurrentUICulture** détermine la culture utilisée par le Gestionnaire de ressources et l'objet **My.Resources**. Elle utilise ces informations pour rechercher des ressources spécifiques à la culture au moment de l'exécution.

Pour récupérer la culture de l'interface utilisateur actuelle, vous pouvez utiliser la propriété My.Application.UICulture ou la propriété **CurrentUICulture**.

Utilisez la méthode **My.Application.ChangeCulture** pour modifier la culture que le thread actuel utilise pour la manipulation et la mise en forme des chaînes.

Exemple

Cet exemple utilise la méthode **My.Application.ChangeUICulture** pour définir la culture que l'objet My.Resources utilise pour récupérer des ressources.

VB

```
Sub ShowLocalizedMessage()
    Dim culture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.Message)
    My.Application.ChangeUICulture(culture)
End Sub
```

Pour que cet exemple fonctionne, votre application doit avoir une chaîne nommée `Message` dans le fichier de ressources de l'application et cette dernière doit avoir la version de la culture française de ce fichier de ressources, `Resources.fr-FR.resx`. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Si l'application n'a pas la version de la culture française de ce fichier de ressources, l'objet **My.Resource** récupère la ressource à partir du fichier de ressources de la culture par défaut.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase, ApplicationBase

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : récupérer des ressources localisées dans Visual Basic

Référence

My.Application, objet

My.Application.ChangeCulture, méthode

My.Resources, objet

Microsoft.VisualBasic.ApplicationServices.ApplicationBase.ChangeUICulture(System.String)

My.Application.Culture, propriété

Obtient la culture utilisée par le thread actuel pour la manipulation et la mise en forme de chaînes.

```
' Usage
Dim value As System.Globalization.CultureInfo = My.Application.Culture
' Declaration
Public ReadOnly Property Culture As System.Globalization.CultureInfo
```

Valeur de retour

Retourne un objet CultureInfo qui représente la culture utilisée par le thread actuel pour la manipulation et la mise en forme de chaînes.

Notes

La propriété **My.Application.CurrentCulture** obtient l'objet **CultureInfo** utilisé par le thread actuel pour la manipulation et la mise en forme de chaînes. Cet objet est identique à celui retourné par la propriété CurrentCulture, qui contrôle plusieurs calculs liés aux chaînes sur ce thread. La propriété **CurrentCulture** détermine les formats par défaut pour les dates, les heures, la monnaie et les chiffres. Elle détermine également le tri, la comparaison et la mise en majuscules des chaînes.

Pour changer la culture, vous pouvez utiliser la méthode My.Application.ChangeCulture ou assigner un objet **CultureInfo** à la propriété **CurrentCulture**.

Le paramètre **CurrentCulture** est différent d'un paramètre de langue. Elle contient uniquement des données liées aux paramètres standard en vigueur dans une région géographique.

Utilisez la propriété **My.Application.CurrentUICulture** pour obtenir la culture utilisée par le thread actuel pour récupérer des ressources propres à la culture.

Exemple

Cet exemple montre comment la culture affecte la représentation sous forme de chaîne des dates.

VB

```
Private Sub TestChangeCulture()
    ' Store the current culture.
    Dim currentculture As String = My.Application.Culture.Name
    MsgBox("Current culture is " & currentculture)

    Dim jan1 As New Date(2005, 1, 1, 15, 15, 15)

    My.Application.ChangeCulture("en-US")
    MsgBox("Date represented in en-US culture: " & jan1)
    ' 1/1/2005 3:15:15 PM

    My.Application.ChangeCulture("")
    MsgBox("Date represented in invariant culture" & jan1)
    ' 01/01/2005 15:15:15

    ' Restore the culture.
    My.Application.ChangeCulture(currentculture)
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase, ApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet
My.Application.ChangeCulture, méthode
My.Application.UICulture, propriété
System.Globalization.CultureInfo
CurrentCulture
Microsoft.VisualBasic.ApplicationServices.ApplicationBase.Culture

My.Application.CommandLineArgs, propriété

Obtient une collection qui contient les arguments de ligne de commande sous forme de chaînes pour l'application actuelle.

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Application.
CommandLineArgs
' Declaration
Public ReadOnly Property CommandLineArgs As System.Collections.ObjectModel.ReadOnlyCollecti
on(Of String)
```

Valeur de retour

ReadOnlyCollection de **String**, qui contient les arguments de ligne de commande sous forme de chaînes pour l'application actuelle.

Notes

La propriété **My.Application.CommandLineArgs** fournit un accès en lecture seule aux arguments de ligne de commande de l'application actuelle pour les applications qui ne sont pas déployées par ClickOnce.

Pour une application à instance unique, la propriété **My.Application.CommandLineArgs** retourne les arguments de ligne de commande pour la première instance d'une application. Pour accéder aux arguments pour les tentatives de démarrage ultérieures d'une application à instance unique, vous devez gérer l'événement My.Application.StartupNextInstance et examiner la propriété CommandLine de l'argument StartupEventArgs.

Remarque

La propriété **My.Application.CommandLineArgs** retourne uniquement les arguments de ligne de commande. Elle diffère du comportement de la propriété CommandLine qui retourne le nom de l'application, en plus des arguments.

Remarque

Dans une application déployée par ClickOnce, utilisez la propriété ActivationUri de l'objet **My.Application.Deployment** pour obtenir les arguments de ligne de commande. Pour plus d'informations, consultez My.Application.Deployment, propriété.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.Application.CommandLineArgs**.

Pour	Consultez
Vérifier si l'application a démarré avec la chaîne `/batch` comme argument	Comment : activer un mode batch pour des applications Windows Forms
Vérifier les arguments de ligne de commande pour les tentatives de démarrage ultérieures d'une application à instance unique	My.Application.StartupNextInstance, événement

Exemple

Cet exemple utilise la propriété **My.Application.CommandLineArgs** pour examiner les arguments de ligne de commande de l'application. Si un argument détecté démarre avec `/input=`, le reste de cet argument est affiché.

VB

```
Private Sub ParseCommandLineArgs()
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In My.Application.CommandLineArgs
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
```

```
        Next

        If inputName = "" Then
            MsgBox("No input name")
        Else
            MsgBox("Input name: " & inputName)
        End If
    End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : ConsoleApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	**Oui**
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
EnvironmentPermission	Contrôle la possibilité d'accéder à la variable d'environnement PATH. Énumération associée : Read.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet
My.Application.StartupNextInstance, événement
ReadOnlyCollection
Microsoft.VisualBasic.ApplicationServices.ConsoleApplicationBase.CommandLineArgs
StartupEventArgs
CommandLine

My.Application.Deployment, propriété

Obtient l'objet de déploiement ClickOnce de l'application actuelle qui fournit une prise en charge pour la mise à jour du déploiement actuel par programme et une prise en charge pour le téléchargement de fichiers à la demande.

```
' Usage
Dim value As System.Deployment.Application.ApplicationDeployment = My.Application.Deploymen
t
' Declaration
Public ReadOnly Property Deployment As System.Deployment.Application.ApplicationDeployment
```

Valeur de retour

Retourne l'objet ApplicationDeployment pour le déploiement ClickOnce de l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- l'application n'est pas déployée comme une application ClickOnce (InvalidDeploymentException).

Notes

La propriété **My.Application.Deployment** retourne l'objet ClickOnce **ApplicationDeployment** de l'application. L'objet **ApplicationDeployment** fournit une prise en charge pour la mise à jour du déploiement actuel par programme ainsi qu'une prise en charge pour le téléchargement de fichiers à la demande. Pour plus d'informations sur les applications ClickOnce et comment les déployer, consultez Déploiement ClickOnce et Publication d'applications ClickOnce.

Vérifiez la propriété My.Application.IsNetworkDeployed avant d'accéder à la propriété **My.Application.Deployment**. Sinon, si l'application n'est pas déployée à l'aide de ClickOnce, la lecture de la propriété **My.Application.Deployment** produit une exception **InvalidDeploymentException**. Pour plus d'informations sur le déploiement d'une application ClickOnce, consultez Comment : publier une application ClickOnce.

Pour plus d'informations sur l'utilisation de la propriété **My.Application.Deployment**, consultez Vue d'ensemble du modèle d'application Visual Basic.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.Application.Deployment**.

Pour	**Consultez**
Rechercher une mise à jour d'application	Comment : rechercher les mises à jour de l'application ClickOnce
Télécharger une mise à jour d'application	Comment : télécharger les mises à jour d'applications ClickOnce

Exemple

Cet exemple télécharge et installe la mise à jour après avoir vérifié que l'application est déployée via le réseau. La méthode Update ne met pas à jour l'application tant qu'elle n'est pas obsolète. Pour pouvoir utiliser la mise à jour, l'application doit redémarrer. Pour plus d'informations, consultez Comment : télécharger les mises à jour d'applications ClickOnce.

VB

```
Sub UpdateApplication()
    If My.Application.IsNetworkDeployed Then
        My.Application.Deployment.Update()
    End If
End Sub
```

Seules les applications qui sont déployées via ClickOnce peuvent être mises à jour à l'aide de l'objet **My.Application.Deployment**. Pour plus d'informations sur le déploiement d'une application ClickOnce, consultez Comment : publier une application ClickOnce.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : ConsoleApplicationBase

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	**Oui**
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : rechercher les mises à jour de l'application ClickOnce
Comment : télécharger les mises à jour d'applications ClickOnce
Comment : publier une application ClickOnce

Référence

My.Application, objet
My.Application.IsNetworkDeployed, propriété
System.Deployment.Application.ApplicationDeployment
Microsoft.VisualBasic.ApplicationServices.ConsoleApplicationBase.Deployment

Concepts

Vue d'ensemble du modèle d'application Visual Basic

Autres ressources

Publication d'applications ClickOnce

My.Application.DoEvents, méthode

Traite tous les messages Windows qui sont actuellement contenus dans la file d'attente de messages.

```
' Usage
My.Application.DoEvents()
' Declaration
Public Sub DoEvents()
```

Notes

La méthode **My.Application.DoEvents** permet à votre application de gérer d'autres événements qui peuvent être déclenchés durant l'exécution de votre code. La méthode **My.Application.DoEvents** a le même comportement que la méthode DoEvents.

Lorsque vous exécutez une application Windows Forms, elle crée un nouveau formulaire qui attend ensuite les événements à gérer. Chaque fois que le formulaire gère un événement, par exemple un clic de bouton, il traite l'ensemble du code associé à cet événement. Tous les autres événements attendent dans la file d'attente. Lorsque votre code gère l'événement, votre application ne répond pas. Par exemple, la fenêtre ne se repeint pas si une autre fenêtre est glissée par-dessus.

Si vous appelez **My.Application.DoEvents** dans votre code, votre application peut gérer les autres événements. Par exemple, si votre code ajoute des données à ListBox dans une boucle, et s'il appelle **My.Application.DoEvents** après chaque étape de la boucle, votre formulaire se repeint lorsqu'une autre fenêtre est glissée par-dessus. Si vous supprimez **My.Application.DoEvents** de votre code, votre formulaire ne se repeint qu'à la fin de l'exécution du gestionnaire d'événements Click du bouton.

En général, vous utilisez cette méthode dans une boucle pour traiter des messages.

Remarque

La méthode **My.Application.DoEvents** ne traite pas les événements exactement de la même manière que le formulaire. Utilisez le multithreading pour permettre au formulaire de gérer directement les événements. Pour plus d'informations, consultez Multithreading dans Visual Basic.

Attention

Si une méthode qui gère un événement d'interface utilisateur appelle la méthode **My.Application.DoEvents**, celle-ci peut être entrée à nouveau avant de terminer. Cela peut se produire car la méthode **My.Application.DoEvents** traite des messages Windows, et ce type de message peut déclencher des événements.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Application.DoEvents**.

Pour	Consultez
Autoriser un formulaire à répondre à une entrée d'interface utilisateur lorsqu'il est occupé	Procédure pas à pas : gestion des événements

Exemple

Cet exemple utilise la méthode **My.Application.DoEvents** pour permettre la mise à jour de l'interface utilisateur de `TextBox1`.

VB

```
Private Sub TestDoEvents()
    For i As Integer = 0 To 10000
        TextBox1.Text = i.ToString
        My.Application.DoEvents()
    Next
End Sub
```

Ce code doit être présent dans un formulaire qui contient un composant `TextBox1` avec une propriété `Text`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : AllWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.DoEvents

DoEvents

My.Application.GetEnvironmentVariable, méthode

Retourne la valeur de la variable d'environnement spécifiée.

```
' Usage
Dim value As String = My.Application.GetEnvironmentVariable(name)
' Declaration
Public Function GetEnvironmentVariable( _
   ByVal name As String _
) As String
```

Paramètres

name

String qui contient le nom de la variable d'environnement.

Valeur de retour

String qui contient la valeur de la variable d'environnement portant le nom *name*.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- L'argument *name* a la valeur **Nothing** (ArgumentNullException).
- La variable d'environnement spécifiée par *name* n'existe pas (ArgumentException).
- Le code appelant ne dispose pas de EnvironmentPermission avec un accès **Read** (SecurityException).

Notes

La méthode **My.Application.GetEnvironmentVariable** retourne la variable d'environnement portant le nom *name*. Cette méthode est semblable à System.Environment.GetEnvironmentVariable(System.String), sauf que celle-ci lève une exception si la variable d'environnement spécifiée par *name* n'existe pas.

Exemple

Cet exemple utilise la méthode **My.Application.GetEnvironmentVariable** pour obtenir et afficher la valeur de la variable d'environnement PATH, si elle est disponible. Sinon, elle affiche un message qui indique que la variable d'environnement PATH n'existe pas.

VB

```
Private Sub TestGetEnvironmentVariable()
    Try
        MsgBox("PATH = " & My.Application.GetEnvironmentVariable("PATH"))
    Catch ex As System.ArgumentException
        MsgBox("Environment variable 'PATH' does not exist.")
    End Try
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase, ApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**

Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet

GetEnvironmentVariable

GetEnvironmentVariable

My.Application.Info, propriété

Obtient un objet qui fournit des propriétés pour accéder aux informations sur l'assembly de l'application, par exemple le numéro de version, la description, etc.

```
' Usage
Dim value As Microsoft.VisualBasic.ApplicationServices.AssemblyInfo = My.Application.Info
' Declaration
Public ReadOnly Property Info As AssemblyInfo
```

Valeur de retour

Cette propriété retourne l'objet **My.Application.Info** pour l'application actuelle.

Notes

Cette propriété fournit un accès facile à **My.Application.Info**. Pour plus d'informations, consultez My.Application.Info, objet.

Exemple

Cet exemple utilise la propriété **My.Application.Info.Version** pour afficher la version de l'application.

VB

```
MsgBox("Application version: " & My.Application.Info.Version.ToString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : ApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet

My.Application.Info, objet

My.Application.IsNetworkDeployed, propriété

Obtient un **Boolean** qui indique si l'application a été déployée à partir d'un réseau utilisant ClickOnce.

```
' Usage
Dim value As Boolean = My.Application.IsNetworkDeployed
' Declaration
Public ReadOnly Property IsNetworkDeployed As Boolean
```

Valeur de retour

Boolean qui indique si l'application a été déployée à partir d'un réseau. La valeur est **True** si l'application actuelle a été déployée à partir d'un réseau ; sinon, elle est **False**.

Notes

La propriété **My.Application.IsNetworkDeployed** indique si l'application a été déployée à partir d'un réseau utilisant ClickOnce. Pour plus d'informations sur les applications ClickOnce, consultez Déploiement ClickOnce.

Vous devez vérifier que cette propriété a la valeur **True** avant d'accéder à la propriété My.Application.Deployment. Sinon, si l'application n'est pas déployée à partir d'un réseau utilisant ClickOnce, la lecture de la propriété **My.Application.Deployment** produit une exception InvalidDeploymentException.

Pour plus d'informations sur la configuration de ClickOnce, consultez Publication d'applications ClickOnce. Pour plus d'informations sur le déploiement d'une application ClickOnce, consultez Comment : publier une application ClickOnce.

Pour plus d'informations sur l'utilisation de la propriété **My.Application.IsNetworkDeployed**, consultez Vue d'ensemble du modèle d'application Visual Basic.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.Application.IsNetworkDeployed**.

Pour	**Consultez**
Rechercher une mise à jour d'application.	Comment : rechercher les mises à jour de l'application ClickOnce
Télécharger une mise à jour d'application.	Comment : télécharger les mises à jour d'applications ClickOnce

Exemple

Cet exemple télécharge et installe la mise à jour après avoir vérifié que l'application est déployée à partir d'un réseau. La méthode Update ne met pas à jour l'application tant qu'elle n'est pas obsolète. Pour pouvoir utiliser la mise à jour, l'application doit redémarrer. Pour plus d'informations, consultez Comment : télécharger les mises à jour d'applications ClickOnce.

VB

```
Sub UpdateApplication()
    If My.Application.IsNetworkDeployed Then
        My.Application.Deployment.Update()
    End If
End Sub
```

Seules les applications qui sont déployées via ClickOnce peuvent être mises à jour à l'aide de l'objet **My.Application.Deployment**. Pour plus d'informations sur le déploiement d'une application ClickOnce, consultez Comment : publier une application ClickOnce.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : ConsoleApplicationBase

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	**Oui**
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : rechercher les mises à jour de l'application ClickOnce

Comment : télécharger les mises à jour d'applications ClickOnce

Comment : publier une application ClickOnce

Référence

My.Application, objet

My.Application.Deployment, propriété

IsNetworkDeployed

Concepts

Vue d'ensemble du modèle d'application Visual Basic

My.Application.Log, propriété

Obtient un objet qui fournit des propriétés et des méthodes pour écrire des informations sur les événements et les exceptions dans les écouteurs de journalisation de l'application.

```
' Usage
Dim value As Microsoft.VisualBasic.Logging.Log = My.Application.Log
' Declaration
Public ReadOnly Property Log As MyLog
```

Valeur de retour

Cette propriété retourne l'objet **My.Application.Log** pour l'application actuelle.

Notes

Cette propriété fournit un accès facile à l'objet **My.Application.Log**. Pour plus d'informations, consultez My.Application.Log, objet.

Exemple

Cet exemple montre comment utiliser la méthode **My.Application.Log.WriteEntry** pour enregistrer des informations de traçage. Pour plus d'informations, consultez Comment : écrire des messages de journal.

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : ApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet

My.Application.Log, objet

My.Application.MinimumSplashScreenDisplayTime, propriété

Détermine la durée minimale, en millisecondes, pendant laquelle l'écran de démarrage s'affiche.

```
' Usage
Dim value As Integer = My.Application.MinimumSplashScreenDisplayTime
' Declaration
Public Property MinimumSplashScreenDisplayTime As Integer
```

Valeur de retour

Integer. Durée minimale, en millisecondes, pendant laquelle l'écran de démarrage s'affiche.

Notes

La propriété **My.Application.MinimumSplashScreenDisplayTime** vous permet de spécifier la durée minimale pendant laquelle l'écran de démarrage de l'application s'affiche. Si le formulaire principal termine l'initialisation dans un délai inférieur à la valeur spécifiée par cette propriété, l'écran de démarrage reste affiché jusqu'à l'expiration de la durée demandée, puis le formulaire principal s'affiche. Si votre application met plus de temps à démarrer, l'écran de démarrage est occulté une fois que le formulaire principal devient actif.

Lorsque vous ajoutez un écran de démarrage à votre application à l'aide du Concepteur de projets, il affecte 2000 à la propriété **My.Application.MinimumSplashScreenDisplayTime**, soit une durée d'affichage minimale de deux secondes.

Cette propriété prend en charge le modèle d'application Visual Basic. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Vous devez définir la propriété **My.Application.MinimumSplashScreenDisplayTime** dans une méthode qui substitue la méthode OnInitialize ou OnCreateSplashScreen. Le code qui substitue les méthodes de la classe WindowsFormsApplicationBase doit être entré dans le fichier ApplicationEvents.vb qui est masqué par défaut.

Pour accéder à la fenêtre de l'éditeur de code afin de substituer des membres

1. Un projet étant sélectionné dans l'**Explorateur de solutions**, cliquez dans le menu **Projet** sur **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Exemple

Cet exemple montre comment définir la propriété **My.Application.MinimumSplashScreenDisplayTime** en substituant la propriété **OnInitialize**.

VB

```
Protected Overrides Function OnInitialize( _
    ByVal commandLineArgs As _
    System.Collections.ObjectModel.ReadOnlyCollection(Of String) _
) As Boolean
    ' Set the display time to 5000 milliseconds (5 seconds).
    Me.MinimumSplashScreenDisplayTime = 5000
    Return MyBase.OnInitialize(commandLineArgs)
End Function
```

Cet exemple exige que le projet possède un écran de démarrage.

Vous devez entrer ce code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Autorisations

Aucune autorisation n'est requise.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Voir aussi

Référence

My.Application, objet
My.Application.SplashScreen, propriété
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.MinimumSplashScreenDisplayTime
OnInitialize
OnCreateSplashScreen
WindowsFormsApplicationBase

Concepts

Extension du modèle d'application Visual Basic

My.Application.NetworkAvailabilityChanged, événement

Se produit en cas de modification de la disponibilité du réseau.

```
' Usage
Public Sub Me_NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As NetworkAvailableEventArgs _
) Handles Me.NetworkAvailabilityChanged
End Sub
' Declaration
Public Event NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As NetworkAvailableEventArgs _
)
```

Paramètres

sender

Obligatoire. Objet **Object** qui a déclenché l'événement.

e

Obligatoire. Objet NetworkAvailableEventArgs qui contient des informations sur la disponibilité du réseau.

Notes

Une application déclenche l'événement **NetworkAvailabilityChanged** lorsque la disponibilité du réseau change. Vous pouvez utiliser la propriété IsNetworkAvailable du paramètre *e* pour obtenir le nouvel état de la connexion réseau. Pour obtenir l'état actuel de la connexion réseau, utilisez la propriété My.Computer.Network.IsAvailable.

Cet événement est déclenché sur le thread principal de l'application avec les autres événements de l'interface utilisateur. Cela permet au gestionnaire d'événements d'accéder directement à l'interface utilisateur de l'application. Toutefois, si l'application est occupée à gérer un autre événement de l'interface utilisateur lorsque cet événement est déclenché, ce dernier ne peut pas être traité tant que l'autre gestionnaire d'événements n'a pas fini ou appelé la méthode My.Application.DoEvents.

L'événement My.Computer.Network.NetworkAvailabilityChanged fournit les mêmes fonctionnalités que cet événement, mais est disponible pour tous les types d'application.

Le code du gestionnaire d'événements **NetworkAvailabilityChanged** est stocké dans le fichier ApplicationEvents.vb, qui est masqué par défaut.

Pour accéder à la fenêtre de l'éditeur de code pour les événements d'application

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Cliquez sur **Propriétés** dans le menu **Projet**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Remarque

De nombreux concentrateurs de réseau fournissent une connexion réseau même si le concentrateur est déconnecté d'un plus grand réseau. Par conséquent, pour les connexions câblées, cet événement indique la modification de la connexion entre l'ordinateur et un concentrateur.

Remarque

L'événement **NetworkAvailabilityChanged** n'est pas déclenché par les applications qui s'exécutent sur Windows 95 et Windows 98, ou qui sont exécutées par une personne autre qu'un administrateur sur Windows 2000. Si votre application s'exécute sur ces plates-formes, utilisez lapropriété My.Computer.Network.IsAvailable pour vérifier la disponibilité du réseau.

Exemple

Cet exemple appelle la méthode `SetConnectionStatus` de la classe `Form1` par défaut lorsque la disponibilité du réseau change.

Vous devez entrer le code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour accéder à cette fenêtre, suivez les instructions contenues dans la section Notes de cette rubrique.

VB

```
Private Sub MyApplication_NetworkAvailabilityChanged( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs _
) Handles Me.NetworkAvailabilityChanged
    My.Forms.Form1.SetConnectionStatus(e.IsNetworkAvailable)
End Sub
```

Compilation du code

Le projet doit contenir un formulaire nommé `Form1`. Il doit contenir une méthode nommée `SetConnectionStatus` qui accepte un paramètre **Boolean**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet
My.Forms, objet
My.Computer.Network.NetworkAvailabilityChanged, événement
Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs

My.Application.OpenForms, propriété

Obtient une collection de tous les formulaires ouverts de l'application.

```
' Usage
Dim value As System.Windows.Forms.FormCollection = My.Application.OpenForms
' Declaration
Public ReadOnly Property OpenForms As System.Windows.Forms.FormCollection
```

Valeur de retour

Objet FormCollection qui contient tous les formulaires ouverts de l'application.

Notes

La propriété **My.Application.OpenForms** obtient une collection de tous les formulaires ouverts de l'application. Le comportement est identique à la propriété System.Windows.Forms.Application.OpenForms.

Remarque

La propriété **My.Application.OpenForms** retourne tous les formulaires ouverts quel que soit le thread qui les a ouverts. Vo us devez vérifier la propriété InvokeRequired de chaque formulaire avant d'y accéder ; sinon, elle peut lever une exception InvalidOperationException. Pour plus d'informations, consultez Comment : accéder à tous les formulaires ouverts d'une application.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.Application.OpenForms**.

Pour	Reportez-vous à
Afficher les titres de tous les formulaires ouverts de l'appli cation	Comment : accéder à tous les formulaires ouverts d'une application

Exemple

Cet exemple effectue une boucle sur les formulaires ouverts de l'application, sélectionne ceux qui sont directement accessibles par le thread actuel et affiche leurs titres dans un contrôle ListBox. Pour accéder aux formulaires ouverts, consultez Comment : accéder à tous les formulaires ouverts d'une application.

VB

```
Private Sub GetOpenFormTitles()
    Dim formTitles As New Collection

    Try
        For Each f As Form In My.Application.OpenForms
            If Not f.InvokeRequired Then
                ' Can access the form directly.
                formTitles.Add(f.Text)
            End If
        Next
    Catch ex As Exception
        formTitles.Add("Error: " & ex.Message)
    End Try

    Form1.ListBox1.DataSource = formTitles
End Sub
```

Cet exemple requiert que votre application Windows Forms ait un formulaire nommé `Form1` contenant une zone de liste nommée `ListBox1`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au Presse-papiers. Énumération associée : AllWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet
System.Windows.Forms.FormCollection
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.OpenForms
System.Windows.Forms.Application.OpenForms

My.Application.Run, méthode

Configure et démarre le modèle d'application Visual Basic.

```
' Usage
My.Application.Run(commandLine)
' Declaration
Public Sub Run( _
   ByVal commandLine As String() _
)
```

Paramètres

commandLine

Tableau de type **String**. Ligne de commande de **Sub Main**.

Notes

La méthode **My.Application.Run** prend en charge le modèle d'application Visual Basic. Cette méthode doit être appelée uniquement à partir du **Sub Main** de l'application. Par défaut, le **Sub Main** d'une application Windows Forms appelle cette méthode. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
EnvironmentPermission	Contrôle la possibilité d'accéder à toutes les variables d'environnement. Énumération associée : Unrestricted.
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : **Unrestricted**.
RegistryPermission	Contrôle la possibilité d'accéder aux variables du Registre. Énumération associée : **Unrestricted**.

UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : AllWindows.
WebPermission	Contrôle les droits d'accès aux ressources Internet HTTP. Énumération associée : **Unrestricted**.
SocketPermission	Contrôle les droits de création ou d'acception de connexions sur une adresse de transport. Énumération associée : **Unrestricted**.
PerformanceCounterPermission	Contrôle l'accès à un composant de compteur de performance Windows NT. Énumération associée : **Unrestricted**.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Run(System.String[])

Concepts

Vue d'ensemble du modèle d'application Visual Basic

My.Application.SaveMySettingsOnExit, propriété

Détermine si l'application enregistre les paramètres utilisateur en quittant.

```
' Usage
Dim value As Boolean = My.Application.SaveMySettingsOnExit
' Declaration
Public Property SaveMySettingsOnExit As Boolean
```

Valeur de retour

Boolean. **True** indique que l'application enregistre les paramètres utilisateur en quittant. Sinon, les paramètres ne sont pas enregistrés implicitement.

Notes

La propriété **SaveMySettingsOnExit** permet à l'utilisateur de modifier le mode d'enregistrement des paramètres par l'application, au moment de l'exécution. Vous pouvez enregistrer explicitement les modifications de paramètre à l'aide de la méthode Save de l'objet **My.Settings**. Pour plus d'informations sur la modification et l'enregistrement de paramètres, consultez My.Settings, objet.

Les modifications apportées à cette propriété ne sont pas rendues persistantes lors de la fermeture de l'application. Pour changer la propriété **SaveMySettingsOnExit** de manière permanente, vous devez modifier le paramètre dans le **Concepteur de projets** :

Pour modifier le paramètre dans le Concepteur de projets

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Cliquez sur **Propriétés** dans le menu **Projet**.
2. Cliquez sur l'onglet **Application**.
3. Sélectionnez **Enregistrer My.Settings lors de l'arrêt**.

Pour plus d'informations, consultez Gestion des propriétés de l'application.

Exemple

Cet exemple se compose de deux parties.

- Sous-routine `InitializeSaveMySettingsOnExit`, qui initialise l'état activé d'un contrôle CheckBox sur la valeur actuelle de la propriété **SaveMySettingsOnExit**.
- Sous-routine `SaveMySettingsOnExit_CheckedChanged`, qui gère les modifications apportées à un contrôle **CheckBox** pour mettre à jour la propriété **My.Application.SaveMySettingsOnExit**.

VB

```
Private Sub InitializeSaveMySettingsOnExit()
    SaveMySettingsOnExit.Checked = _
        My.Application.SaveMySettingsOnExit
End Sub
Private Sub SaveMySettingsOnExit_CheckedChanged( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs _
) Handles SaveMySettingsOnExit.CheckedChanged
    My.Application.SaveMySettingsOnExit = _
        SaveMySettingsOnExit.Checked
End Sub
```

Pour que cet exemple fonctionne, votre application doit contenir un contrôle **CheckBox** nommé `SaveMySettingsOnExit`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet
My.Settings, objet
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.SaveMySettingsOnExit
Save

My.Application.Shutdown, événement

Se produit à l'arrêt de l'application.

```
' Usage
Public Sub Me_Shutdown( _
   ByVal sender As Object, _
   ByVal e As System.EventArgs _
) Handles Me.Shutdown
End Sub
' Declaration
Public Event Shutdown( _
   ByVal sender As Object, _
   ByVal e As System.EventArgs _
)
```

Paramètres

sender

Objet **Object** qui a déclenché l'événement.

e

Objet EventArgs qui contient Empty.

Notes

Une application déclenche l'événement **Shutdown** avant de quitter. Cela vous permet de contrôler la manière dont l'application ferme ses ressources. Cet événement fait partie du modèle d'application Visual Basic. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Le code du gestionnaire d'événements **Shutdown** est stocké dans le fichier ApplicationEvents.vb, qui est masqué par défaut.

Pour accéder à la fenêtre de l'éditeur de code pour les événements d'application

1. Un projet étant sélectionné dans l'**Explorateur de solutions**, cliquez dans le menu **Projet** sur **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'événement **My.Application.Shutdown**.

Pour	Consultez
Utiliser les événements fournis par le modèle d 'application Visual Basic pour exécuter le code	Comment : exécuter le code lorsque l'application démarre ou se termine
Enregistrer un message lors de l'arrêt de l'appli cation	Comment : enregistrer des messages lorsque l'application démarre ou s'arrête

Exemple

Cet exemple enregistre des messages lors de l'arrêt de l'application.

VB

```
Private Sub MyApplication_Shutdown( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Me.Shutdown
    My.Application.Log.WriteEntry("Application Shut Down.")
```

```
End Sub
```

Vous devez entrer le code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour accéder à cette fenêtre, suivez les instructions contenues dans la section Notes de cette rubrique. Pour plus d'informations, consultez Comment : enregistrer des messages lorsque l'application démarre ou s'arrête.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : enregistrer des messages lorsque l'application démarre ou s'arrête

Comment : gérer les événements d'application (Visual Basic)

Référence

My.Application, objet

System.EventArgs

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Shutdown

Concepts

Vue d'ensemble du modèle d'application Visual Basic

My.Application.SplashScreen, propriété

Obtient ou définit l'écran de démarrage pour cette application.

```
' Usage
Dim value As System.Windows.Forms.Form = My.Application.SplashScreen
' Declaration
Public Property SplashScreen As System.Windows.Forms.Form
```

Valeur de retour

Objet Form que l'application utilise comme écran de démarrage.

Exceptions

La condition ci-dessous peut générer une exception :

- La même valeur est assignée à cette propriété et à la propriété **My.Application.MainForm** (ArgumentNullException).

Notes

La propriété **My.Application.SplashScreen** vous permet d'obtenir ou de définir l'objet **Form** que l'application utilise comme écran de démarrage, c'est-à-dire le premier formulaire graphique qui s'affiche au démarrage d'une application.

Cette propriété prend en charge le modèle d'application Visual Basic. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Les modifications apportées à cette propriété ne sont pas rendues persistantes lors de la fermeture de l'application. Pour changer l'écran de démarrage de manière permanente, vous devez modifier le paramètre dans le **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier un écran de démarrage pour une application (Visual Basic).

Exemple

Cet exemple utilise la propriété **My.Application.SplashScreen** et l'événement **My.Application.Startup** pour mettre à jour l'écran de démarrage à l'aide des informations d'état au démarrage de l'application.

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    ' Get the splash screen.
    Dim splash As SplashScreen1 = CType(My.Application.SplashScreen, SplashScreen1)
    ' Display current status information.
    splash.Status = "Current user: " & My.User.Name
End Sub
```

Cet exemple requiert que le projet possède un écran de démarrage nommé `SplashScreen1`. L'écran de démarrage doit contenir une propriété nommée `Status` qui met à jour son interface utilisateur.

Vous devez entrer ce code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible

Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application, objet

My.Application.MinimumSplashScreenDisplayTime, propriété

System.Windows.Forms.Form

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.SplashScreen

My.Application.Startup, événement

Se produit au démarrage de l'application.

```
' Usage
Public Sub Me_Startup( _
   ByVal sender As Object, _
   ByVal e As StartupEventArgs _
) Handles Me.Startup
End Sub
' Declaration
Public Event Startup( _
   ByVal sender As Object, _
   ByVal e As StartupEventArgs _
)
```

Paramètres

sender

Objet **Object** qui a déclenché l'événement.

e

Objet StartupEventArgs qui contient les arguments de ligne de commande de l'application.

Notes

Une application (à instance non unique) normale déclenche l'événement **Startup** à chaque démarrage. Une application à instance unique déclenche l'événement **Startup** au démarrage uniquement si l'application n'est pas déjà active ; sinon, elle déclenche l'événement **StartupNextInstance**. Pour plus d'informations, consultez My.Application.StartupNextInstance, événement et Comment : spécifier le comportement d'instanciation pour une application.

Cet événement fait partie du modèle d'application Visual Basic. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Vous pouvez utiliser la propriété Cancel du paramètre *e* pour contrôler le chargement du formulaire de démarrage d'une application. Lorsque la propriété **Cancel** a la valeur **True**, le formulaire de démarrage ne démarre pas. Dans ce cas, votre code doit appeler un autre chemin d'accès de code de démarrage. Par exemple, consultez Comment : activer un mode batch pour des applications Windows Forms.

Vous pouvez utiliser la propriété CommandLine du paramètre *e* ou la propriété My.Application.CommandLineArgs pour accéder aux arguments de ligne de commande de l'application.

Le code du gestionnaire d'événements **Startup** est stocké dans le fichier ApplicationEvents.vb, qui est masqué par défaut.

Pour accéder à la fenêtre de l'éditeur de code pour les événements d'application

1. Un projet étant sélectionné dans l'**Explorateur de solutions**, cliquez dans le menu **Projet** sur **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'événement **My.Application.Startup**.

Pour	Consultez
Utiliser les événements fournis par le modèle d'application Visual Basic pour exécuter le code	Comment : exécuter le code lorsque l'application démarre ou se termine

Vérifier si l'application a démarré avec la chaîne `/batch` comme argument	Comment : activer un mode batch pour des applications Windows Forms

Exemple

Cet exemple utilise la propriété **My.Application.SplashScreen** et l'événement **My.Application.Startup** pour mettre à jour l'écran de démarrage à l'aide des informations d'état au démarrage de l'application.

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    ' Get the splash screen.
    Dim splash As SplashScreen1 = CType(My.Application.SplashScreen, SplashScreen1)
    ' Display current status information.
    splash.Status = "Current user: " & My.User.Name
End Sub
```

Cet exemple requiert que le projet possède un écran de démarrage nommé `SplashScreen1`. L'écran de démarrage doit contenir une propriété nommée `Status` qui met à jour son interface utilisateur.

Vous devez entrer le code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour accéder à cette fenêtre, suivez les instructions contenues dans la section Notes de cette rubrique. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
SecurityPermission	Contrôle la capacité d'ajouter un gestionnaire d'événements pour cet événement. Énumération associée : System.Security.Permissions.SecurityPermissionFlag.ControlAppDomain.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : activer un mode batch pour des applications Windows Forms
Comment : gérer les événements d'application (Visual Basic)
Comment : spécifier le comportement d'instanciation pour une application

Référence

My.Application, objet
My.Application.StartupNextInstance, événement
My.Application.CommandLineArgs, propriété
Microsoft.VisualBasic.ApplicationServices.StartupEventArgs
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Startup

Concepts

Vue d'ensemble du modèle d'application Visual Basic

My.Application.StartupNextInstance, événement

Se produit lors de la tentative de démarrer une application à instance unique alors que l'application est déjà active.

```
' Usage
Public Sub Me_StartupNextInstance( _
   ByVal sender As Object, _
   ByVal e As StartupNextInstanceEventArgs _
) Handles Me.StartupNextInstance
End Sub
' Declaration
Public Event StartupNextInstance( _
   ByVal sender As Object, _
   ByVal e As StartupNextInstanceEventArgs _
)
```

Paramètres

sender

Objet **Object** qui a déclenché l'événement.

e

Objet StartupEventArgs qui contient les arguments de ligne de commande de l'application.

Notes

Une application à instance unique déclenche l'événement StartupNextInstance lorsque vous essayez de redémarrer l'application alors qu'elle est déjà active. Lorsqu'une application à instance unique démarre pour la première fois, elle déclenche l'événement Startup. Pour plus d'informations, consultez My.Application.Startup, événement et Comment : spécifier le comportement d'instanciation pour une application.

Cet événement fait partie du modèle d'application Visual Basic. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Cet événement est déclenché sur le thread principal de l'application avec les autres événements de l'interface utilisateur. Ainsi, le gestionnaire d'événements peut accéder directement à l'interface utilisateur de l'application. Toutefois, si l'application est occupée par la gestion d'un autre événement de l'interface utilisateur lorsque cet événement est déclenché, ce dernier ne pourra pas être traité tant que l'autre gestionnaire d'événements n'a pas terminé ou appelé la méthode My.Application.DoEvents, méthode.

Remarque

L'événement **StartupNextInstance** est déclenché uniquement dans les applications à instance unique. Pour activer le comportement à instance unique pour votre application, vous devez activer la case à cocher **Application à instance unique** dans le Concepteur de projets. Pour plus d'informations, consultez Comment : spécifier le comportement d'instanciation pour une application.

Vous devez utiliser la propriété CommandLine du paramètre *e* afin d'accéder aux arguments pour les tentatives ultérieures de démarrer une application à instance unique. My.Application.CommandLineArgs, propriété fournit les arguments utilisés pour démarrer la première instance d'une application à instance unique.

Le code du gestionnaire d'événements **StartupNextInstance** est stocké dans le fichier ApplicationEvents.vb, qui est masqué par défaut.

Pour accéder à la fenêtre de l'éditeur de code pour les événements d'application

1. Un projet étant sélectionné dans l'**Explorateur de solutions**, cliquez dans le menu **Projet** sur **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

   Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'événement **My.Application.StartupNextInstance**.

Pour	Reportez-vous à
Utiliser les événements fournis par le modèle d'application Visual Basic pour exécuter le code	Comment : exécuter le code lorsque l'application démarre ou se termine
Vérifier les arguments de ligne de commande de la première instance de l'application	My.Application.CommandLineArgs, propriété

Exemple

Cet exemple utilise le paramètre *e* du gestionnaire d'événements **StartupNextInstance** pour examiner les arguments de ligne de commande de l'application. Si un argument commençant par `/input=` est trouvé, le reste de cet argument est affiché.

VB

```
Private Sub MyApplication_StartupNextInstance( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs _
) Handles Me.StartupNextInstance
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In e.CommandLine
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next

    If inputName = "" Then
        MsgBox("No input name")
    Else
        MsgBox("Input name: " & inputName)
    End If
End Sub
```

Vous devez entrer le code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour accéder à cette fenêtre, suivez les instructions de la section Remarques de cette rubrique. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	Oui
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non

Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : gérer les événements d'application (Visual Basic)

Comment : spécifier le comportement d'instanciation pour une application

Comment : spécifier le comportement d'instanciation pour une application

Référence

My.Application, objet

My.Application.Startup, événement

My.Application.CommandLineArgs, propriété

StartupEventArgs

CommandLine

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.StartupNextInstance

Concepts

Vue d'ensemble du modèle d'application Visual Basic

My.Application.UICulture, propriété

Obtient la culture utilisée par le thread actuel pour récupérer des ressources propres à la culture.

```
' Usage
Dim value As System.Globalization.CultureInfo = My.Application.UICulture
' Declaration
Public ReadOnly Property UICulture As System.Globalization.CultureInfo
```

Valeur de retour

Retourne un objet CultureInfo qui représente la culture utilisée par le thread actuel pour récupérer des ressources propres à la culture.

Notes

La propriété **My.Application.CurrentUICulture** obtient l'objet **CultureInfo** utilisé par le thread actuel pour récupérer des ressources propres à la culture. Cet objet est identique à celui retourné par la propriété CurrentUICulture. La propriété **CurrentUICulture** détermine la culture utilisée par le gestionnaire des ressources et l'objet **My.Resources** ; ces informations lui sont utiles pour rechercher des ressources propres à la culture au moment de l'exécution.

Pour modifier la culture, vous pouvez utiliser la méthode My.Application.ChangeUICulture ou assigner un autre objet **CultureInfo** à la propriété **CurrentUICulture**.

Le paramètre CurrentCulture est différent d'un paramètre de langue. Elle contient uniquement des données liées aux paramètres standard en vigueur dans une région géographique.

Utilisez la propriété **My.Application.CurrentCulture** pour obtenir la culture utilisée par le thread actuel pour la manipulation et la mise en forme de chaînes.

Exemple

Cet exemple utilise la propriété **My.Application.CurrentCulture** pour mettre en cache la culture actuelle, avant de la modifier avec la méthode My.Application.ChangeUICulture. La méthode **My.Application.ChangeUICulture** définit la culture utilisée par l'objet My.Resources pour récupérer des ressources.

VB

```
Sub ShowLocalizedMessage()
    Dim culture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.Message)
    My.Application.ChangeUICulture(culture)
End Sub
```

Pour que cet exemple fonctionne, votre application doit contenir une chaîne nommée `Message` dans son fichier de ressources, et elle doit posséder la version de la culture française de ce fichier de ressources, `Resources.fr-FR.resx`. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Si l'application ne dispose pas de la version de la culture française de ce fichier de ressources, l'objet **My.Resource** récupère la ressource du fichier de ressources de la culture par défaut.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase, ApplicationBase

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**

Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : récupérer des ressources localisées dans Visual Basic

Référence

My.Application, objet

My.Application.ChangeUICulture, méthode

My.Application.Culture, propriété

System.Globalization.CultureInfo

CurrentUICulture

Microsoft.VisualBasic.ApplicationServices.ApplicationBase.UICulture

My.Application.UnhandledException, événement

Se produit lorsque l'application rencontre une exception non gérée.

```
' Usage
Public Sub Me_UnhandledException( _
   ByVal sender As Object, _
   ByVal e As UnhandledExceptionEventArgs _
) Handles Me.UnhandledException
End Sub
' Declaration
Public Event UnhandledException( _
   ByVal sender As Object, _
   ByVal e As UnhandledExceptionEventArgs _
)
```

Paramètres

sender

Objet **Object** qui a déclenché l'événement.

e

Objet UnhandledExceptionEventArgs qui contient l'exception non gérée et des informations supplémentaires.

Notes

Une application déclenche l'événement **UnhandledException** lorsqu'elle rencontre une exception non gérée. Cet événement fait partie du modèle d'application Visual Basic. Pour plus d'informations, consultez Vue d'ensemble du modèle d'application Visual Basic.

Vous pouvez utiliser la propriété Exception du paramètre *e* pour accéder à l'exception non gérée qui a provoqué cet événement.

Vous pouvez utiliser la propriété ExitApplication du paramètre *e* pour contrôler la fermeture de l'application. Étant donné que **ExitApplication** a la valeur **True** par défaut, l'application se ferme après l'exécution du gestionnaire d'événements **UnhandledException**. Vous pouvez lui affecter **True** dans le gestionnaire d'événements **UnhandledException** pour que l'application continue à fonctionner, et la rétablir dans un état d'attente.

Le code du gestionnaire d'événements **UnhandledException** est stocké dans le fichier ApplicationEvents.vb, qui est masqué par défaut.

Pour accéder à la fenêtre de l'éditeur de code pour les événements d'application :

1. Un projet étant sélectionné dans l'**Explorateur de solutions**, cliquez dans le menu **Projet** sur **Propriétés**.
2. Cliquez sur l'onglet **Application**.
3. Cliquez sur le bouton **Afficher les événements de l'application** pour ouvrir l'éditeur de code.

Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Remarque

Le compilateur Visual Basic empêche les applications générées pour le débogage de déclencher cet événement, afin de perm ettre à un débogueur de gérer les exceptions non gérées. Cela signifie que si vous testez votre application en l'exécutant dans le débogueur Environnement de développement intégré de Visual Studio, votre gestionnaire d'événements **UnhandledExce ption** ne sera pas appelé. Pour plus d'informations sur la génération d'applications pour le débogage, consultez /debug (Visual Basic).

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'événement **My.Application.UnhandledException**.

Pour	Consultez

Utiliser les événements fournis par le modèle d'application Visual Basic pour exécuter le code	Comment : exécuter le code lorsque l'application démarre ou se termine
Enregistrer les exceptions non gérées	Comment : enregistrer des exceptions dans Visual Basic

Exemple

Cet exemple utilise l'événement **My.Application.UnhandledException** pour enregistrer les exceptions non gérées.

VB

```
Private Sub MyApplication_UnhandledException( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.UnhandledExceptionEventArgs _
) Handles Me.UnhandledException
    My.Application.Log.WriteException(e.Exception, _
        TraceEventType.Critical, _
        "Unhandled Exception.")
End Sub
```

Vous devez entrer le code dans la fenêtre de l'éditeur de code pour les événements d'application. Pour accéder à cette fenêtre, suivez la procédure indiquée dans la section Notes de cette rubrique. Pour plus d'informations, consultez Comment : gérer les événements d'application (Visual Basic).

Étant donné que l'événement **UnhandledException** n'est pas déclenché lorsqu'un débogueur est associé à l'application, vous devez exécuter cet exemple en dehors du débogueur Environnement de développement intégré de Visual Studio.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : WindowsFormsApplicationBase

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
SecurityPermission	Contrôle la capacité d'ajouter un gestionnaire d'événements pour cet événement. Énumération associée : System.Security.Permissions.SecurityPermissionFlag.ControlAppDomain.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application, objet

Microsoft.VisualBasic.ApplicationServices.UnhandledExceptionEventArgs

Concepts

Vue d'ensemble du modèle d'application Visual Basic

My.Application.Info, objet

Fournit des propriétés pour accéder aux informations sur l'application, par exemple le numéro de version, la description, les assemblys chargés, etc.

Notes

Certaines propriétés exposées par l'objet **My.Application.Info** retournent les informations sur l'assembly de l'application. Pour définir les informations dans l'assembly de l'application, vous devez utiliser la boîte de dialogue **Informations de l'assembly**, qui est accessible dans le volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

L'objet **My.Application.Info** fournit des fonctionnalités semblables à la classe FileVersionInfo, mais sa portée est définie pour permettre un accès rapide aux propriétés les plus communes.

Les autres propriétés, telles que **LoadedAssemblies**, fournissent un accès à l'état actuel de l'application.

Exemple

Cet exemple utilise la propriété **My.Application.Info.Version** pour afficher la version de l'application.

VB

```
MsgBox("Application version: " & My.Application.Info.Version.ToString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de l'objet My.Application.Info
My.Application, objet
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo
FileVersionInfo

Membres de l'objet My.Application.Info

L'objet My.Application.Info fournit des propriétés pour accéder aux informations sur l'application, par exemple le numéro de version, la description, les assemblys chargés, etc.

Propriétés

Propriété	Description
AssemblyName	Obtient le nom, sans extension, du fichier de l'assembly pour l'application.
CompanyName	Obtient le nom de société associé à l'application.
Copyright	Obtient le texte de copyright associé à l'application.
Description	Obtient la description associée à l'application.
DirectoryPath	Obtient le répertoire où l'application est stockée.
LoadedAssemblies	Obtient une collection de tous les assemblys chargés par l'application.
ProductName	Obtient le nom de produit associé à l'application.
StackTrace	Obtient les informations de traçage de la pile actuelles.
Title	Obtient le titre associé à l'application.
Marque	Obtient le texte de marque associé à l'application.
Version	Obtient le numéro de version de l'application.
WorkingSet	Obtient la quantité de mémoire physique mappée au contexte du processus.

Voir aussi

Référence

My.Application.Info, objet

My.Application.Info.AssemblyName, propriété

Obtient le nom, sans l'extension, du fichier d'assembly de l'application.

```
' Usage
Dim value As String = My.Application.Info.AssemblyName
' Declaration
Public ReadOnly Property AssemblyName As String
```

Valeur de retour

String contenant le nom du fichier.

Notes

La propriété **My.Application.Info.AssemblyName** obtient le nom du fichier contenant le manifeste.

Exemple

Cet exemple utilise la propriété **My.Application.Info.AssemblyName** pour afficher le nom de l'application.

VB

```
MsgBox("Application assembly name: " & My.Application.Info.AssemblyName)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application.Info, objet AssemblyName

My.Application.Info.CompanyName, propriété

Obtient le nom de société associé à l'application.

```
' Usage
Dim value As String = My.Application.Info.CompanyName
' Declaration
Public ReadOnly Property CompanyName As String
```

Valeur de retour

String qui contient le nom de société associé à l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- L'assembly ne possède pas de AssemblyCompanyAttribute (InvalidOperationException).

Notes

La propriété **My.Application.Info.CompanyName** obtient le nom de société associé à l'application.

Vous pouvez changer le nom de société en modifiant la valeur de **Société** dans la boîte de dialogue **Informations de l'assembly**, qui est accessible dans le volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple utilise la propriété **My.Application.Info.CompanyName** pour afficher le nom de société associé à l'application.

VB

```
MsgBox("Application created by: " & My.Application.Info.CompanyName)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.CompanyName

My.Application.Info.Copyright, propriété

Obtient une mention de droits d'auteur associée à l'application.

```
' Usage
Dim value As String = My.Application.Info.Copyright
' Declaration
Public ReadOnly Property Copyright As String
```

Valeur de retour

String contenant la mention de droits d'auteur associée à l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- l'assembly ne possède pas de AssemblyCopyrightAttribute (InvalidOperationException).

Notes

La propriété **My.Application.Info.CompanyName** obtient la mention de droits d'auteur associée à l'application.

Vous pouvez modifier la mention des droits d'auteur en modifiant la valeur correspondant à **Copyright** dans la boîte de dialogue **Informations de l'assembly**, accessible à partir du volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple utilise la propriété **My.Application.Info.CompanyName** pour afficher le nom de la société associé à l'application.

VB

```
MsgBox("Application copyright: " & My.Application.Info.Copyright)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Copyright

My.Application.Info.Description, propriété

Obtient la description associée à l'application.

```
' Usage
Dim value As String = My.Application.Info.Description
' Declaration
Public ReadOnly Property Description As String
```

Valeur de retour

String qui contient la description associée à l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- L'assembly ne possède pas de AssemblyDescriptionAttribute (InvalidOperationException).

Notes

La propriété **My.Application.Info.Description** obtient la description associée à l'application.

Vous pouvez changer la description en modifiant la valeur de **Description** dans la boîte de dialogue **Informations de l'assembly**, qui est accessible dans le volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple utilise la propriété **My.Application.Info.Description** pour afficher la description associée à l'application.

VB

```
MsgBox("Application description: " & My.Application.Info.Description)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Description

My.Application.Info.DirectoryPath, propriété

Obtient le répertoire où l'application est stockée.

```
' Usage
Dim value As String = My.Application.Info.DirectoryPath
' Declaration
Public ReadOnly Property DirectoryPath As String
```

Valeur de retour

String qui contient le répertoire où l'application est stockée.

Notes

La propriété **My.Application.Info.DirectoryPath** obtient le répertoire où l'application est stockée.

Exemple

Cet exemple utilise la propriété **My.Application.Info.DirectoryPath** pour afficher le chemin d'accès du répertoire où l'application est stockée.

VB

```
MsgBox("Application directory path: " & My.Application.Info.DirectoryPath)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.DirectoryPath

My.Application.Info.LoadedAssemblies, propriété

Obtient une collection de tous les assemblys chargés par l'application.

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of System.Reflection.Assembl
y) = My.Application.Info.LoadedAssemblies
' Declaration
Public ReadOnly Property LoadedAssemblies As System.Collections.ObjectModel.ReadOnlyCollect
ion(Of System.Reflection.Assembly)
```

Valeur de retour

ReadOnlyCollection de Assembly qui contient tous les assemblys chargés par l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- Le domaine d'application n'est pas chargé (AppDomainUnloadedException).

Notes

La propriété **My.Application.Info.LoadedAssemblies** est semblable à la méthode GetAssemblies.

Exemple

Cet exemple utilise la propriété **My.Application.Info.LoadedAssemblies** pour afficher les assemblys chargés par l'application.

VB

```
ListBox1.DataSource = My.Application.Info.LoadedAssemblies
```

Cet exemple requiert que votre application Windows Forms contienne un contrôle ListBox nommé `ListBox1`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Application.Info, objet

ReadOnlyCollection

System.Reflection.Assembly

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.LoadedAssemblies

My.Application.Info.ProductName, propriété

Obtient le nom de produit associé à l'application.

```
' Usage
Dim value As String = My.Application.Info.ProductName
' Declaration
Public ReadOnly Property ProductName As String
```

Valeur de retour

String contenant le nom de produit associé à l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- L'assembly ne possède pas de AssemblyProductAttribute (InvalidOperationException).

Notes

La propriété **My.Application.Info.ProductName** obtient le nom de produit associé à l'application.

Vous pouvez changer le nom du produit en modifiant la valeur correspondant à **Produit** dans la boîte de dialogue **Informations de l'assembly**, accessible à partir du volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple utilise la propriété **My.Application.Info.ProductName** pour afficher le nom de produit associé à l'application.

VB

```
MsgBox("Application product name: " & My.Application.Info.ProductName)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.ProductName

My.Application.Info.StackTrace, propriété

Obtient les informations de traçage de la pile actuelles.

```
' Usage
Dim value As String = My.Application.Info.StackTrace
' Declaration
Public ReadOnly Property StackTrace As String
```

Valeur de retour

String qui contient les informations de traçage de la pile actuelles. La valeur de retour peut être Empty.

Exceptions

La condition ci-dessous peut générer une exception :

- Les informations de traçage de la pile demandées sont hors limites (ArgumentOutOfRangeException).

Notes

La propriété **My.Application.Info.StackTrace** répertorie les appels de méthode dans l'ordre chronologique inverse, c'est-à-dire elle décrit d'abord l'appel de méthode le plus récent, puis elle répertorie chaque appel de méthode de la pile sur une ligne disctincte. Toutefois, la propriété **My.Application.Info.StackTrace** peut ne pas indiquer autant d'appels de méthode que prévu, en raison des transformations de code qui se produisent pendant l'optimisation.

En général, les informations de traçage de la pile sont utilisées pour déboguer des applications.

Les informations de traçage de la pile pour chaque appel de méthode sont mises en forme comme suit :

"à *FullClassName*.*MethodName*(*MethodParams*) dans *FileName*:ligne *LineNumber*"

Le littéral "à" est précédé de trois espaces, et les littéraux "dans" et ":ligne" sont omis si les symboles de débogage ne sont pas disponibles. Les espaces réservés *FullClassName*, *MethodName*, *MethodParams*, *FileName* et *LineNumber* sont remplacés par des valeurs réelles définies comme suit :

Élément	Description
FullClassName	Nom complet de la classe, y compris l'espace de noms.
MethodName	Nom de la méthode.
MethodParams	Liste des paires type/nom de paramètre. Chaque paire est séparée par une virgule (,). Ces informations sont omises si *MethodName* n'accepte pas de paramètres.
FileName	Nom du fichier source où la méthode *MethodName* est déclarée. Ces informations sont omises si les symboles de débogage ne sont pas disponibles.
LineNumber	Numéro de la ligne dans *FileName* qui contient le code source de *MethodName* pour l'instruction présente sur la pile des appels. Ces informations sont omises si les symboles de débogage ne sont pas disponibles.

La propriété **My.Applicaiton.Info.StackTrace** fournit des fonctionnalités semblables à la propriété StackTrace.

Exemple

Cet exemple utilise la propriété **My.Application.Info.StackTrace** pour afficher la trace de la pile de l'application à partir de laquelle le code s'exécute.

VB

```
MsgBox("Stack trace: " & My.Application.Info.StackTrace)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder aux informations dans les chemins d'accès. Énumération associée : PathDiscovery.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application.Info, objet

StackTrace

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.StackTrace

My.Application.Info.Title, propriété

Obtient le titre associé à l'application.

```
' Usage
Dim value As String = My.Application.Info.Title
' Declaration
Public ReadOnly Property Title As String
```

Valeur de retour

String qui contient l'AssemblyTitleAttribute associé à l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- L'assembly ne possède pas de **AssemblyTitleAttribute** (InvalidOperationException).

Notes

La propriété **My.Application.Info.Title** obtient le titre associé à l'application.

Vous pouvez changer le titre en modifiant la valeur de **Titre** dans la boîte de dialogue **Informations de l'assembly**, qui est accessible dans le volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple utilise la propriété **My.Application.Info.Title** pour afficher le titre associé à l'application.

VB

```
MsgBox("Application title: " & My.Application.Info.Title)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Title

My.Application.Info.Trademark, propriété

Obtient le texte de marque associé à l'application.

```
' Usage
Dim value As String = My.Application.Info.Trademark
' Declaration
Public ReadOnly Property Trademark As String
```

Valeur de retour

String qui contient le texte de marque associé à l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- L'assembly ne possède pas de AssemblyTrademarkAttribute (InvalidOperationException).

Notes

La propriété **My.Application.Info.Trademark** obtient le texte de marque associé à l'application.

Vous pouvez changer le texte de marque en modifiant la valeur de **Marque** dans la boîte de dialogue **Informations de l'assembly**, qui est accessible dans le volet **Application** du **Concepteur de projets**. Pour plus d'informations, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple utilise la propriété **My.Application.Info.Trademark** pour afficher les informations de marque associées à l'application.

VB

```
MsgBox("Application trademark: " & My.Application.Info.Trademark)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : spécifier des informations d'assembly

Référence

My.Application.Info, objet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Trademark

My.Application.Info.Version, propriété

Obtient le numéro de version de l'application.

```
' Usage
Dim value As System.Version = My.Application.Info.Version
' Declaration
Public ReadOnly Property Version As System.Version
```

Valeur de retour

Objet Version contenant le numéro de version de l'application.

Exceptions

La condition ci-dessous peut générer une exception :

- L'application n'a pas les autorisations suffisantes pour accéder à la version de l'assembly (SecurityException).

Notes

La propriété **My.Application.Info.Version** obtient un objet **Version** contenant le numéro de version de l'application. Vous pouvez utiliser les propriétés Major, Minor, Build et Revision de l'objet **Version** pour accéder aux informations de version spécifiques concernant l'application.

Les applications déployées par ClickOnce doivent utiliser la propriété CurrentVersion de la propriété My.Application.Deployment.

Exemple

Cet exemple utilise la propriété **My.Application.Info.Version** pour afficher la version de l'application.

VB

```
MsgBox("Application version: " & My.Application.Info.Version.ToString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application.Info, objet
My.Application.Deployment, propriété
CurrentVersion
System.Version
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Version

My.Application.Info.WorkingSet, propriété

Obtient la quantité de mémoire physique mappée au contexte du processus.

```
' Usage
Dim value As Long = My.Application.Info.WorkingSet
' Declaration
Public ReadOnly Property WorkingSet As Long
```

Valeur de retour

Long qui contient le nombre d'octets de mémoire physique mappée au contexte du processus.

Exceptions

La condition ci-dessous peut générer une exception :

- Il existe une situation de confiance partielle et l'utilisateur ne dispose des autorisations nécessaires (SecurityException).

Notes

La propriété **My.Application.Info.WorkingSet** a le même comportement que la propriété WorkingSet.

Remarque
Cette propriété retourne toujours zéro pour Windows 98 et Windows Millennium.

Exemple

Cet exemple utilise la propriété **My.Application.Info.WorkingSet** pour afficher le nombre d'octets de mémoire physique mappée à l'application.

VB

```
MsgBox("Application working set: " & My.Application.Info.WorkingSet)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : AssemblyInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
EnvironmentPermission	Contrôle la possibilité d'accéder à toutes les variables d'environnement. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Application.Info, objet

WorkingSet

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.WorkingSet

My.Application.Log, objet

Fournit une propriété et des méthodes pour écrire des informations sur les événements et les exceptions dans les écouteurs de journalisation de l'application.

Notes

L'objet **My.Application.Log** fournit un point d'entrée simple à partir duquel accéder aux services d'enregistrement du .NET Framework. Les méthodes **WriteEntry** et **WriteException** écrivent des messages dans les écouteurs de journalisation de l'application. Les écouteurs peuvent être configurés par le fichier de configuration de l'application. Pour plus d'informations, consultez Procédure pas à pas : modification de l'emplacement des informations My.Application.Log et Utilisation des journaux d'applications dans Visual Basic.

L'objet **My.Application.Log** est uniquement disponible pour les applications clientes. Pour les applications Web, utilisez **My.Log**. Pour plus d'informations, consultez My.Log, objet.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Application.Log**.

Pour	Consultez
Écrire des informations sur les événements dans les écouteurs de journalisation de l'application	Comment : écrire des messages de journal
Écrire des informations sur les exceptions dans les écouteurs de journalisation de l'application	Comment : enregistrer des exceptions dans Visual Basic
Déterminer l'emplacement où **My.Application.Log** écrit les informations	Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Exemple

Cet exemple montre comment utiliser la méthode **My.Application.Log.WriteEntry** pour enregistrer des informations de traçage. Pour plus d'informations, consultez Comment : écrire des messages de journal.

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Logging

Classe : Log

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**

Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Voir aussi

Référence

Membres de l'objet My.Application.Log

My.Application, objet

Membres de l'objet My.Application.Log

L'objet My.Application.Log fournit une propriété et des méthodes pour écrire des informations sur les événements et les exceptions dans les écouteurs de journalisation de l'application.

Propriétés

Propriété	Description
TraceSource	Accède à l'objet TraceSource sous-jacent de l'objet **My.Application.Log**. Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Méthodes

Méthode	Description
WriteEntry	Écrit un message dans les écouteurs de journalisation de l'application.
WriteException	Écrit des informations sur les exceptions dans les écouteurs de journalisation de l'application.

Voir aussi

Référence

My.Application.Log, objet

DefaultFileLogWriter, propriété (My.Application.Log et My.Log)

Obtient l'objet FileLogTraceListener que l'objet **Log** utilise.

```
' Usage
Dim value As Logging.FileLogTraceListener = My.Application.Log.DefaultFileLogWriter
' Web usage
Dim value As Logging.FileLogTraceListener = My.Log.DefaultFileLogWriter
' Declaration
Public ReadOnly Property DefaultFileLogWriter As Logging.FileLogTraceListener
```

Valeur de retour

Retourne l'objet **FileLogTraceListener** que l'objet **Log** utilise.

Notes

Vous pouvez utiliser cette propriété pour déterminer la configuration actuelle de l'objet **FileLogTraceListener** de l'objet **Log**.

Dans les applications clientes, l'objet **Log** est disponible par le biais de l'objet **My.Application.Log**. Dans les applications Web, l'objet **Log** est disponible par le biais de l'objet **My.Log**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Logging

Classe : Log

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Dépannage : écouteurs de journalisation

Référence

My.Application.Log, objet

My.Log, objet

FileLogTraceListener

TraceSource, propriété (My.Application.Log et My.Log)

Accède à l'objet TraceSource sous-jacent de l'objet **Log**.

```
' Usage
Dim value As System.Diagnostics.TraceSource = My.Application.Log.TraceSource
' Web usage
Dim value As System.Diagnostics.TraceSource = My.Log.TraceSource
' Declaration
Public ReadOnly Property TraceSource As System.Diagnostics.TraceSource
```

Valeur de retour

Retourne l'objet **TraceSource** sous-jacent de l'objet **Log**.

Notes

Vous pouvez utiliser cette propriété pour déterminer la configuration actuelle de l'objet **Log**.

Dans les applications clientes, l'objet **Log** est disponible par l'intermédiaire de l'objet **My.Application.Log**. Dans les applications Web, l'objet **Log** est disponible par l'intermédiaire de l'objet **My.Log**.

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous sélectionnez l'onglet **Tous**.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la propriété **TraceSource**.

Pour	**Consultez**
Obtenir des informations pour chacun des écouteurs de journalisation de l'objet **Log**.	Dépannage : écouteurs de journalisation

Configuration requise

Espace de noms : Microsoft.VisualBasic.Logging

Classe : Log

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : écouteurs de journalisation

Référence

My.Application.Log, objet

My.Log, objet

System.Diagnostics.TraceSource

Microsoft.VisualBasic.Logging.Log.TraceSource

WriteEntry, méthode (My.Application.Log et My.Log)

Écrit un message dans les écouteurs de journalisation de l'application.

```
' Usage
My.Application.Log.WriteEntry(message)
My.Application.Log.WriteEntry(message ,severity)
My.Application.Log.WriteEntry(message ,severity ,id)
' Web usage
My.Log.WriteEntry(message)
My.Log.WriteEntry(message ,severity)
My.Log.WriteEntry(message ,severity ,id)
' Declaration
Public Sub WriteEntry( _
   ByVal message As String _
)
' -or-
Public Sub WriteEntry( _
   ByVal message As String, _
   ByVal severity As System.Diagnostics.TraceEventType _
)
' -or-
Public Sub WriteEntry( _
   ByVal message As String, _
   ByVal severity As System.Diagnostics.TraceEventType, _
   ByVal id As Integer _
)
```

Paramètres

message

Obligatoire. Message à enregistrer. Si *message* a la valeur **Nothing**, une chaîne vide est utilisée.

severity

Type de message. Par défaut, **TraceEventType.Information**.

id

Identificateur de message, généralement utilisé pour la corrélation. Par défaut, associé à *entryType*, comme décrit dans le tableau.

Exceptions

Les conditions ci-dessous génèrent une exception :

- Le type d'exception ou de message est **Nothing** (ArgumentNullException).
- Le type de message ne fait pas partie des valeurs de l'énumération TraceEventType (InvalidEnumArgumentException).
- Un code avec une confiance partielle appelle la méthode, mais écrit dans un écouteur de journalisation des événements qui requiert une confiance totale (SecurityException).

Notes

La méthode **WriteEntry** écrit un message dans les écouteurs de journalisation des événements de l'application.

Dans les applications clientes, l'objet **Log** est disponible par l'intermédiaire de l'objet **My.Application.Log**. Dans les applications Web, l'objet **Log** est disponible par l'intermédiaire de l'objet **My.Log**.

Pour déterminer les écouteurs de journalisation qui reçoivent les messages de la méthode **WriteEntry**, consultez Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log. Vous pouvez modifier des écouteurs de journalisation par défaut. Pour plus d'informations, consultez Utilisation des journaux d'applications dans Visual Basic.

Pour les surcharges qui n'acceptent pas l'argument *id*, l'*id* écrit dans le journal est défini par le tableau suivant.

severity	*id* par défaut
Information	0
Warning	1
Error	2
Critical	3
Start	4
Stop	5
Suspend	6
Resume	7
Verbose	8
Transfer	9

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **WriteEntry**.

Pour	Consultez
Écrire des informations sur les événements dans les écouteurs de journalisation de l'application	Comment : écrire des messages de journal
Déterminer l'emplacement où **Log** écrit les informations	Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Exemple

Cet exemple montre comment utiliser la méthode **My.Application.Log.WriteEntry** pour enregistrer des informations de traçage. Pour plus d'informations, consultez Comment : écrire des messages de journal.

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

Cet exemple de code ne peut s'exécuter que dans une application cliente. Remplacez **My.Application.Log.WriteEntry** par **My.Log.WriteEntry** pour les applications Web.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Logging

Classe : Log

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : écrire des messages de journal

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet

My.Log, objet

System.Diagnostics.TraceEventType

Microsoft.VisualBasic.Logging.Log.WriteEntry(System.String)

Concepts

Utilisation des journaux d'applications dans Visual Basic

WriteException, méthode (My.Application.Log et My.Log)

Écrit les informations sur les exceptions dans les écouteurs de journalisation de l'application.

```
' Usage
My.Application.Log.WriteException(ex)
My.Application.Log.WriteException(ex ,severity ,additionalInfo)
My.Application.Log.WriteException(ex ,severity ,additionalInfo ,id)
' Web usage
My.Log.WriteException(ex)
My.Log.WriteException(ex ,severity ,additionalInfo)
My.Log.WriteException(ex ,severity ,additionalInfo ,id)
' Declaration
Public Sub WriteException( _
   ByVal ex As System.Exception _
)
' -or-
Public Sub WriteException( _
   ByVal ex As System.Exception, _
   ByVal severity As System.Diagnostics.TraceEventType, _
   ByVal additionalInfo As String _
)
' -or-
Public Sub WriteException( _
   ByVal ex As System.Exception, _
   ByVal severity As System.Diagnostics.TraceEventType, _
   ByVal additionalInfo As String, _
   ByVal id As Integer _
)
```

Paramètres

ex

Obligatoire. Exception à enregistrer.

severity

Type de message. Par défaut, Error.

additionalInfo

Chaîne à ajouter au message. Par défaut, il s'agit d'une chaîne vide.

id

Identificateur de message, utilisé en général pour la corrélation. Par défaut, relatif à *entryType* comme décrit dans le tableau de la section Remarques.

Exceptions

Les conditions ci-dessous généreront une exception :

- l'exception ou le type de message est **Nothing** (ArgumentNullException) ;
- le type de message ne fait pas partie des valeurs de l'énumération TraceEventType (InvalidEnumArgumentException) ;
- le code d'un niveau de confiance partiel appelle la méthode, mais effectue l'écriture dans un écouteur de journal des événements qui requiert une confiance totale (SecurityException).

Notes

La méthode **WriteException** écrit les informations contenues dans une exception dans les écouteurs des journaux des événements de l'application.

Dans les applications clientes, l'objet **Log** est disponible dans l'objet **My.Application.Log**. Dans les applications Web, l'objet **Log** est disponible dans l'objet **My.Log**.

Pour connaître les écouteurs de journalisation qui reçoivent les messages de la méthode **WriteException**, consultez Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log. Vous pouvez modifier des écouteurs de journalisation par défaut. Pour plus d'informations, consultez Utilisation des journaux d'applications dans Visual Basic.

Pour les surcharges qui ne prennent pas l'argument *id*, l'identificateur écrit dans le journal est défini par le tableau suivant.

severity	***id* par défaut**
Information	0
Warning	1
Error	2
Critical	3
Start	4
Stop	5
Suspend	6
Resume	7
Verbose	8
Transfer	9

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **WriteException**.

Pour	**Reportez-vous à**
Écrire les informations sur les exceptions dans les écouteurs des journaux des événements de l'application.	Comment : enregistrer des exceptions dans Visual Basic
Déterminer à quel emplacement **Log** écrit les informations.	Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Exemple

Cet exemple indique comment utiliser la méthode **My.Application.Log.WriteException** pour enregistrer les exceptions. Supprimez les marques de commentaire des lignes `Dim` et `MsgBox` pour provoquer une exception NullReferenceException. Pour plus d'informations, consultez Comment : enregistrer des exceptions dans Visual Basic.

VB

```
Public Sub ExceptionLogTest(ByVal fileName As String)
    Try
        ' Code that might generate an exception goes here.
        ' For example:
        '    Dim x As Object
        '    MsgBox(x.ToString)
    Catch ex As Exception
        My.Application.Log.WriteException(ex, _
            TraceEventType.Error, _
            "Exception in ExceptionLogTest " & _
            "with argument " & fileName & ".")
    End Try
End Sub
```

Cet exemple de code ne peut s'exécuter que dans une application cliente. Remplacez **My.Application.Log.WriteException** par **My.Log.WriteException** pour les applications Web.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Logging

Classe : Log

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : enregistrer des exceptions dans Visual Basic

Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet

My.Log, objet

System.Exception

System.Diagnostics.TraceEventType

Microsoft.VisualBasic.Logging.Log.WriteException(System.Exception)

Concepts

Utilisation des journaux d'applications dans Visual Basic

My.Computer, objet

Fournit des propriétés permettant de manipuler des composants informatiques, tels que le son, l'horloge, le clavier, le système de fichiers, etc.

Notes

Les propriétés exposées par l'objet **My.Computer** retournent des informations concernant l'ordinateur sur lequel l'application est déployée, comme spécifié au moment de l'exécution. En général, ces données diffèrent de celles qui étaient disponibles sur l'ordinateur de développement.

Certains membres, tels que l'objet **My.Computer.Audio**, ne sont disponibles que pour les applications non serveur.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer**.

Pour	**Reportez-vous à**
Vérifier l'état de la connexion	Comment : vérifier l'état de la connexion dans Visual Basic
Déterminer si un ordinateur distant est disponible	Comment : déterminer si un ordinateur distant est disponible dans Visual Basic
Télécharger un fichier	Comment : télécharger un fichier dans Visual Basic
Transférer un fichier	Comment : transférer un fichier dans Visual Basic
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Effacer le Presse-papiers	Comment : effacer le Presse-papiers en Visual Basic
Lire à partir du Presse-Papiers.	Comment : lire le Presse-papiers en Visual Basic
Déterminer ce qui est stocké dans le Presse-papiers	Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic
Enregistrer de l'audio dans le Presse-papiers	Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic
Obtenir une image à partir du Presse-papiers	Comment : récupérer une image du Presse-papiers en Visual Basic

Exemple

Cet exemple utilise la propriété **My.Computer.Name** pour afficher le nom de l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Computer name: " & My.Computer.Name)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer (la classe de base ServerComputer fournit les membres disponibles dans tous les projets)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Computer, membres de l'objet

My.Computer.Audio, objet
My.Computer.Clipboard, objet
My.Computer.Clock, objet
My.Computer.FileSystem, objet
My.Computer.Info, objet
My.Computer.Keyboard, objet
My.Computer.Mouse, objet
My.Computer.Network, objet
My.Computer.Ports, objet
My.Computer.Registry, objet
Microsoft.VisualBasic.Devices.Computer
Microsoft.VisualBasic.Devices.ServerComputer

My.Computer, membres de l'objet

L'objet My.Computer fournit des propriétés permettant de manipuler des composants informatiques, tels que le son, l'horloge, le clavier, le système de fichiers, etc.

Propriétés

Propriété	Description
Audio	Retourne l'objet My.Computer.Audio, qui fournit un accès au système audio de l'ordinateur et fournit des méthodes pour lire les fichiers .wav. Cet objet n'est disponible que pour les applications non-serveurs.
Presse-papiers	Retourne l'objet My.Computer.Clipboard, qui fournit des méthodes pour manipuler le Presse-papiers. Cet objet n'est disponible que pour les applications non-serveurs.
Horloge	Retourne l'objet My.Computer.Clock, qui fournit des propriétés pour accéder à l'heure locale actuelle et au temps UTC (Universal Coordinated Time), équivalent à l'heure GMT (Greenwich Mean Time), à partir de l'horloge système.
FileSystem	Retourne l'objet My.Computer.FileSystem, qui fournit des propriétés et des méthodes pour utiliser les lecteurs, les fichiers et les répertoires.
Info	Retourne l'objet My.Computer.Info, qui fournit des propriétés pour obtenir des informations concernant la mémoire, les assemblys chargés, le nom et le système d'exploitation de l'ordinateur.
Clavier	Retourne l'objet My.Computer.Keyboard, qui fournit des propriétés pour accéder à l'état actuel du clavier, par exemple pour savoir quelles touches sont actuellement utilisées, et fournit une méthode pour envoyer des séquences de touches à la fenêtre active. Cet objet n'est disponible que pour les applications non-serveurs.
Souris	Retourne l'objet My.Computer.Mouse, qui fournit des propriétés pour accéder aux informations sur le format et la configuration de la souris installée sur l'ordinateur local. Cet objet n'est disponible que pour les applications non-serveurs.
Nom	Récupère le nom de l'ordinateur.
Réseau	Retourne l'objet My.Computer.Network, qui peut être utilisé pour accéder aux types de réseau et aux événements.
Ports	Retourne l'objet My.Computer.Ports, qui fournit une propriété et une méthode pour accéder aux ports série de l'ordinateur. Cet objet n'est disponible que pour les applications non-serveurs.
Registre	Retourne l'objet My.Computer.Registry, qui peut être utilisé pour lire et écrire dans le Registre.
Screen	Obtient l'objet Screen qui représente l'écran d'affichage principal de l'ordinateur. Cette propriété est disponible uniquement pour les applications non-serveurs.

Voir aussi

Référence

My.Computer, objet

My.Computer.Audio, propriété

Obtient un objet qui fournit des propriétés pour les méthodes de lecture des sons.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Audio = My.Computer.Audio
' Declaration
Public ReadOnly Property Audio As MyAudio
```

Valeur de retour

Cette propriété retourne l'objet **My.Computer.Audio** pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet **My.Computer.Audio**. Pour plus d'informations, consultez My.Computer.Audio, objet.

Exemple

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **AudioPlayMode.Background** est spécifié.

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

Ce code ne peut s'exécuter que dans une application Windows Forms.

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers audio, pensez à stocker les fichiers comme ressources de l'application. Les fichiers sont ensuite accessibles via l'objet My.Resources.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Audio, objet

My.Computer.Clipboard, propriété

Obtient un objet qui fournit des méthodes pour manipuler le Presse-papiers.

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.ClipboardProxy = My.Computer.Clipboard
' Declaration
Public ReadOnly Property Clipboard As ClipboardProxy
```

Valeur de retour

Cette propriété retourne l'objet My.Computer.Clipboard pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet My.Computer.Clipboard.

Cette propriété est disponible uniquement pour les applications non-serveurs.

Exemple

Cet exemple lit le texte du Presse-papiers dans la chaîne `textOnClipboard`.

VB

```
Dim textOnClipboard As String = My.Computer.Clipboard.GetText()
```

Cet exemple ne fonctionne pas s'il n'y a pas de texte dans le Presse-papiers.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Clipboard, objet

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clock, propriété

Obtient un objet qui fournit des propriétés pour accéder à l'heure locale actuelle et au temps UTC (Universal Coordinated Time), équivalent à l'heure GMT (Greenwich Mean Time), à partir de l'horloge système.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Clock = My.Computer.Clock
' Declaration
Public ReadOnly Property Clock As MyClock
```

Valeur de retour

Cette propriété retourne l'objet **My.Computer.Clock** pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet **My.Computer.Clock**. Pour plus d'informations, consultez My.Computer.Clock, objet.

Exemple

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **AudioPlayMode.Background** est spécifié.

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

Ce code ne peut s'exécuter que dans une application Windows Forms.

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers audio, pensez à stocker les fichiers comme ressources de l'application. Vous pouvez ensuite accéder aux fichiers par l'intermédiaire de l'objet My.Resources.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer, ServerComputer

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Clock, objet

My.Computer.FileSystem, propriété

Obtient un objet qui fournit des propriétés et des méthodes pour utiliser les lecteurs, les fichiers et les répertoires.

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.FileSystemProxy = My.Computer.FileSystem
' Declaration
Public ReadOnly Property FileSystem As FileSystemProxy
```

Valeur de retour

Cette propriété retourne l'objet My.Computer.FileSystem pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet My.Computer.FileSystem.

Exemple

Cet exemple détermine si le dossier `C:\backup\logs` existe et vérifie ses propriétés.

VB

```
Dim logInfo As System.IO.DirectoryInfo
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    logInfo = My.Computer.FileSystem.GetDirectoryInfo _
    ("C:\backup\logs")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer, ServerComputer

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.FileSystem, objet

Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic
Création, suppression et déplacement de fichiers et de répertoires en Visual Basic
Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.Info, propriété

Obtient un objet qui fournit des propriétés pour obtenir des informations concernant la mémoire, les assemblys chargés, le nom et le système d'exploitation de l'ordinateur.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.ComputerInfo = My.Computer.Info
' Declaration
Public ReadOnly Property Info As MyComputerInfo
```

Valeur de retour

Cette propriété retourne l'objet **My.Computer.Info** pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet **My.Computer.Info**. Pour plus d'informations, consultez My.Computer.Info, objet.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.AvailablePhysicalMemory** pour afficher la quantité de mémoire physique disponible pour l'ordinateur sur lequel le code s'exécute.

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.AvailablePhysicalMemory)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer, ServerComputer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Info, objet

My.Computer.Keyboard, propriété

Obtient un objet qui fournit des propriétés pour accéder à l'état actuel du clavier, par exemple pour savoir quelles touches sont actuellement utilisées, et fournit une méthode pour envoyer des séquences de touches à la fenêtre active.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Keyboard = My.Computer.Keyboard
' Declaration
Public ReadOnly Property Keyboard As Microsoft.VisualBasic.Devices.Keyboard
```

Valeur de retour

Cette propriété retourne l'objet **My.Computer.Keyboard** pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet **My.Computer.Keyboard**. Pour plus d'informations, consultez My.Computer.Keyboard, objet.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.CtrlKeyDown** pour déterminer si la touche CTRL de l'ordinateur est actuellement utilisée.

VB

```
If My.Computer.Keyboard.CtrlKeyDown Then
    MsgBox("CTRL key down")
Else
    MsgBox("CTRL key up")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Keyboard, objet

My.Computer.Mouse, propriété

Obtient un objet qui fournit des propriétés permettant d'obtenir des informations sur le format et la configuration de la souris installée sur l'ordinateur local.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Mouse = My.Computer.Mouse
' Declaration
Public ReadOnly Property Mouse As Microsoft.VisualBasic.Devices.Mouse
```

Valeur de retour

Cette propriété retourne l'objet **My.Computer.Mouse** pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet **My.Computer.Mouse**. Pour plus d'informations, consultez My.Computer.Mouse, objet.

Cette propriété est disponible uniquement pour les applications non-serveurs.

Exemple

Cet exemple utilise les propriétés **My.Computer.Mouse.WheelExists** et **My.Computer.Mouse.WheelScrollLines** pour déterminer si la souris a une roulette de défilement et le nombre d'éléments à faire défiler lorsque vous la faites pivoter.

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non

Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Mouse, objet

My.Computer.Name, propriété

Récupère le nom de l'ordinateur.

```
' Usage
Dim value As String = My.Computer.Name
' Declaration
Public ReadOnly Property Name As String
```

Valeur de retour

String contenant le nom de l'ordinateur.

Notes

La propriété **My.Computer.Name** fournit des fonctionnalités semblables à la propriété MachineName.

Exemple

Cet exemple utilise la propriété **My.Computer.Name** pour afficher le nom de l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Computer name: " & My.Computer.Name)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer, ServerComputer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet
MachineName
Microsoft.VisualBasic.Devices.ServerComputer.Name

My.Computer.Network, propriété

Obtient un objet qui fournit une propriété et des méthodes permettant d'interagir avec le réseau auquel l'ordinateur est connecté.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Network = My.Computer.Network
' Declaration
Public ReadOnly Property Network As MyNetwork
```

Valeur de retour

Cette propriété retourne l'objet My.Computer.Network pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet My.Computer.Network.

Exemple

Cet exemple transfère le fichier `Order.txt` dans http://www.cohowinery.com/uploads.

VB

```
My.Computer.Network.UploadFile ( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/uploads.aspx")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer, ServerComputer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Network, objet

Autres ressources

Exécution d'opérations de réseau

My.Computer.Ports, propriété

Obtient un objet qui fournit une propriété et une méthode pour accéder aux ports série de l'ordinateur.

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Ports = My.Computer.Ports
' Declaration
Public ReadOnly Property Ports As MyPorts
```

Valeur de retour

Cette propriété retourne l'objet **My.Computer.Ports**.

Notes

Cette propriété fournit un accès facile à l'objet **My.Computer.Ports**. Pour plus d'informations, consultez My.Computer.Ports, objet.

Exemple

Cet exemple montre comment envoyer des chaînes au port série `COM1` de l'ordinateur.

Le bloc **Using** permet à l'application de fermer le port série même s'il génère une exception. Tout le code qui manipule le port série doit apparaître dans ce bloc ou dans un bloc **Try...Catch...Finally** avec un appel à la méthode Close.

La méthode WriteLine envoie les données au port série.

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

Pour plus d'informations, consultez Comment : envoyer des chaînes aux ports série dans Visual Basic.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**

Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Ports, objet

My.Computer.Registry, propriété

Obtient un objet qui fournit des propriétés et des méthodes pour manipuler le Registre.

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.RegistryProxy = My.Computer.Registry
' Declaration
Public ReadOnly Property Registry As RegistryProxy
```

Valeur de retour

Cette propriété retourne l'objet My.Computer.Registry pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet My.Computer.Registry.

Exemple

Cet exemple lit la valeur `Name` depuis `HKEY_CURRENT_USER\Software\MyApp` et l'affiche dans un message.

VB

```
Dim readValue As Object
        readValue = My.Computer.Registry.GetValue _
        ("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
        MsgBox("The value is " & CStr(readValue))
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer, ServerComputer

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer, objet

My.Computer.Registry, objet

Concepts

Tâches courantes relatives au Registre

My.Computer.Screen, propriété

Obtient l'objet Screen qui représente l'écran d'affichage principal de l'ordinateur.

```
' Usage
Dim value As System.Windows.Forms.Screen = My.Computer.Screen
' Declaration
Public ReadOnly Property Screen As System.Windows.Forms.Screen
```

Valeur de retour

Objet **Screen** qui représente l'écran principal de l'ordinateur.

Notes

La propriété **My.Computer.Screen** fournit des fonctionnalités semblables à la propriété PrimaryScreen.

Exemple

Cet exemple utilise la propriété WorkingArea de la propriété **My.Computer.Screen** pour déterminer la zone de travail de l'écran principal de l'ordinateur, puis ajuste le formulaire à la zone de travail.

VB

```
Private Sub EnlargeForm()
    Me.Size = My.Computer.Screen.WorkingArea.Size
    Me.Location = New System.Drawing.Point(0, 0)
End Sub
```

Cet exemple doit s'exécuter dans une application Windows Forms.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Computer

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description

FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer, objet

System.Windows.Forms.Screen

PrimaryScreen

Microsoft.VisualBasic.Devices.Computer.Screen

My.Computer.Audio, objet

Fournit des méthodes pour lire des sons.

Notes

Vous pouvez utiliser les méthodes **My.Computer.Audio.Play** et **My.Computer.Audio.PlaySystemSound** pour lire des fichiers audio .wav et des sons système.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Audio**.

Pour	Consultez
Lire un son en arrière-plan.	Comment : lire des sons dans Visual Basic
Lire un son une fois et attendre la fin de la lecture.	Comment : lire des sons et attendre qu'ils se terminent dans Visual Basic
Lire un son plusieurs fois.	Comment : lire des sons en boucle dans Visual Basic
Arrêter la lecture d'un son en arrière-plan.	Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic

Exemple

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **AudioPlayMode.Background** est spécifié.

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

Ce code ne peut s'exécuter que dans une application Windows Forms.

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers audio, pensez à stocker les fichiers comme ressources de l'application. Vous pouvez ensuite accéder aux fichiers par l'intermédiaire de l'objet My.Resources.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Audio

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**

Site Web	Non

Voir aussi

Référence

My.Computer.Audio, membres de l'objet

My.Computer, objet

My.Computer.Audio, membres de l'objet

L'objet My.Computer.Audio fournit des méthodes pour lire des sons.

Methods

Play	Lit un fichier audio .wav. Cette méthode est disponible uniquement pour les applications non-serveurs.
PlaySystemSound	Lit un son système. Cette méthode est disponible uniquement pour les applications non-serveurs.
Arrêter	Arrête un son qui effectuer la lecture en arrière-plan. Cette méthode est disponible uniquement pour les applications non-serveurs.

Voir aussi

Référence

My.Computer.Audio, objet

My.Computer.Audio.Play, méthode

Lit un fichier audio .wav.

```
' Usage
My.Computer.Audio.Play(location)
My.Computer.Audio.Play(location ,playMode)
My.Computer.Audio.Play(data ,playMode)
My.Computer.Audio.Play(stream ,playMode)
' Declaration
Public Sub Play( _
   ByVal location As String _
)
' -or-
Public Sub Play( _
   ByVal location As String, _
   ByVal playMode As AudioPlayMode _
)
' -or-
Public Sub Play( _
   ByVal data As Byte(), _
   ByVal playMode As AudioPlayMode _
)
' -or-
Public Sub Play( _
   ByVal stream As System.IO.Stream, _
   ByVal playMode As AudioPlayMode _
)
```

Paramètres

location

String qui contient le nom du fichier audio.

data

Tableau **Byte** qui représente le fichier audio.

stream

Tableau Stream qui représente le fichier audio.

playMode

Mode énumération AudioPlayMode pour lire le son. Par défaut, **AudioPlayMode.Background**.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- *data* ou *stream* a la valeur **Nothing**, ou *location* est une chaîne vide (ArgumentNullException).
- L'argument *playMode* ne fait pas partie des valeurs énumération AudioPlayMode (InvalidEnumArgumentException).
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier nommé par *location* (IOException).
- Le chemin d'accès est incorrect dans *location* (DirectoryNotFoundException)
- Le nom du chemin d'accès dans *location* est trop long (PathTooLongException).
- Il existe une situation d'un niveau de confiance partiel dans laquelle l'utilisateur ne dispose des autorisations nécessaires (SecurityException).

Notes

La méthode **Play** lit le fichier audio .wav stocké sous forme de fichier dans *location*, de tableau d'octets dans *data* ou de flux de données dans *stream*.

Si la surcharge qui accepte uniquement le paramètre *location* est utilisée, la méthode **Play** lit le son en arrière-plan. Sinon, le paramètre *playMode* détermine le mode de lecture du son.

playMode	**Description**
AudioPlayMode.Background	Lit le son en arrière-plan. Le code appelant continue à s'exécuter.
AudioPlayMode.BackgroundLoop	Lit le son en arrière-plan jusqu'à ce que la méthode My.Computer.Audio.Stop soit appelée. Le code appelant continue à s'exécuter.
AudioPlayMode.WaitToComplete	Lit le son et attend son exécution avant que le code appelant continue.

L'arrière-plan permet à l'application d'exécuter du code pendant la lecture des sons. Pour plus d'informations, consultez Comment : lire des sons en boucle dans Visual Basic et Comment : lire des sons dans Visual Basic.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Audio.Play**.

Pour	**Consultez**
Lire un son une fois	Comment : lire des sons et attendre qu'ils se terminent dans Visual Basic
Lire un son plusieurs fois	Comment : lire des sons en boucle dans Visual Basic
Lire un son en arrière-plan	Comment : lire des sons dans Visual Basic

Exemple

La méthode **My.Computer.Audio.Play** lit le son spécifié en arrière-plan lorsque **PlayMode.Background** est spécifié.

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

Cet exemple de code ne peut s'exécuter que dans une application Windows Forms.

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers audio, pensez à stocker les fichiers comme ressources de l'application. Vous pouvez ensuite accéder aux fichiers par l'intermédiaire de l'objet My.Resources.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Audio

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**

Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : ControlThread.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : lire des sons et attendre qu'ils se terminent dans Visual Basic
Comment : lire des sons en boucle dans Visual Basic
Comment : lire des sons dans Visual Basic

Référence

My.Computer.Audio, objet
My.Computer.Audio.Stop, méthode
AudioPlayMode, énumération
System.IO.Stream
Microsoft.VisualBasic.Devices.Audio.Play(System.String)

My.Computer.Audio.PlaySystemSound, méthode

Lit un son système.

```
' Usage
My.Computer.Audio.PlaySystemSound(systemSound)
' Declaration
Public Sub PlaySystemSound( _
   ByVal systemSound As System.Media.SystemSound _
)
```

Paramètres

systemSound

Objet SystemSound qui représente le son système à lire.

Exceptions

La condition ci-dessous peut générer une exception :

- *systemSound* a la valeur **Nothing** (ArgumentNullException).

Notes

La méthode **PlaySystemSound** lit le son système décrit par *systemSound* ; le son est lu une fois, en arrière-plan.

La valeur de *systemSound* doit être l'un des membres partagés de la classe SystemSounds :

- Asterisk
- Beep
- Exclamation
- Hand
- Question

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.Audio.PlaySystemSound**.

Pour	Consultez
Lire un son système	Comment : lire des sons système dans Visual Basic

Exemple

Cet exemple utilise la méthode **My.Computer.Audio.PlaySystemSound** pour lire un son système.

VB

```
Sub PlaySystemSound()
    My.Computer.Audio.PlaySystemSound( _
        System.Media.SystemSounds.Asterisk)
End Sub
```

Cet exemple de code ne peut s'exécuter que dans une application Windows Forms.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Audio

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des sons système dans Visual Basic

Référence

My.Computer.Audio, objet
System.Media.SystemSound
SystemSounds
Microsoft.VisualBasic.Devices.Audio.PlaySystemSound(System.Media.SystemSound)

My.Computer.Audio.Stop, méthode

Arrête la lecture d'un son en arrière-plan.

```
' Usage
My.Computer.Audio.Stop()
' Declaration
Public Sub Stop()
```

Notes

La méthode **My.Computer.Audio.Stop** arrête la lecture d'un son en arrière-plan qui a été démarrée par la méthode **My.Computer.Audio.Play**.

L'arrière-plan permet à l'application d'exécuter du code pendant la lecture des sons. Pour plus d'informations, consultez Comment : lire des sons en boucle dans Visual Basic et Comment : lire des sons dans Visual Basic.

En général, lorsqu'une application lit un son en boucle, elle doit arrêter le son à un point précis.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.Audio.Stop**.

Pour	Consultez
Arrêter la lecture d'un son en arrière-plan	Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic

Exemple

Cet exemple utilise la méthode **My.Computer.Audio.Stop** pour arrêter l'application qui lit actuellement le son en arrière-plan ou en boucle et utilise la méthode **My.Computer.Audio.Play** pour lire un son en boucle en arrière-plan.

VB

```
Sub PlayLoopingBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.BackgroundLoop)
End Sub
Sub StopBackgroundSound()
    My.Computer.Audio.Stop()
End Sub
```

Cet exemple de code ne peut s'exécuter que dans une application Windows Forms.

Le nom de fichier doit faire référence à un fichier son .wav présent sur votre système.

Pour simplifier la gestion de vos fichiers audio, pensez à stocker les fichiers comme ressources de l'application. Vous pouvez ensuite accéder à ces fichiers par l'intermédiaire de l'objet My.Resources.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Audio

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**

Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : arrêter la lecture de sons en arrière-plan dans Visual Basic

Référence

My.Computer.Audio, objet

Microsoft.VisualBasic.Devices.Audio.Stop

My.Computer.Clipboard, objet

Fournit des méthodes pour la manipulation du Presse-papiers.

Notes

Les éléments déplacés ou copiés vers le Presse-papiers persistent même après que l'application a été arrêtée.

Les données sur le Presse-papiers peuvent être dans n'importe quel format de données, également appelé format de Presse-papiers. Pour obtenir la liste de formats prédéfinis à utiliser avec le Presse-papiers, consultez DataFormats. Lorsqu'un élément est déplacé ou copié vers le Presse-papiers, les éléments aux autres formats sont effacés. Pour rendre d'autres formats persistants, utilisez DataObject qui copie tout ce qui se trouve sur le Presse-papiers actuel y compris les éléments collés à partir d'autres applications. Placez des données dans le Presse-papiers dans plusieurs formats, afin d'accroître les chances de récupération des données par une application cible dont vous ignorez les exigences de format.

Étant donné que toutes les applications Windows partagent le Presse-papiers du système, le contenu peut être modifié lorsque vous basculez vers une autre application.

Une classe doit être sérialisable pour être placée dans le Presse-papiers. Pour plus d'informations, consultez Sérialisation.

Lors de l'accès à distance au Presse-papiers, une ThreadStateException est levée à moins que le thread accédant ne fonctionne en mode de thread cloisonné (STA, Single-Thread Apartment). Pour résoudre ce problème, affectez `STA` à *ThreadApartmentState*. Pour plus d'informations, consultez STAThreadAttribute.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Clipboard**.

Pour	Consultez
Effacer le Presse-papiers	Comment : effacer le Presse-papiers en Visual Basic
Lire à partir du Presse-Papiers.	Comment : lire le Presse-papiers en Visual Basic
Savoir ce qui est stocké dans le Presse-papiers	Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic
Enregistrer de l'audio dans le Presse-papiers	Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic
Obtenir une image à partir du Presse-papiers	Comment : récupérer une image du Presse-papiers en Visual Basic
Stocker des données dans le Presse-papiers	Comment : écrire dans le Presse-papiers en Visual Basic

Exemple

Cet exemple lit le texte du Presse-papiers dans la chaîne `textOnClipboard`.

VB

```
Dim textOnClipboard As String = My.Computer.Clipboard.GetText()
```

Cet exemple ne fonctionne pas s'il n'y a pas de texte dans le Presse-papiers.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Voir aussi

Référence

My.Computer.Clipboard, membres de l'objet

My.Computer, objet

System.Windows.Forms.Clipboard

My.Computer.Clipboard, membres de l'objet

L'objet My.Computer.Clipboard fournit différentes méthodes d'utilisation du Presse-papiers.

Methods

Méthode	Description
Clear	Efface le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
ContainsAudio	Détermine si le Presse-papiers contient les données audio. Cette méthode est disponible uniquement pour les applications non-serveurs.
ContainsData	Détermine si le Presse-papiers contient des données au format personnalisé spécifié. Cette méthode est disponible uniquement pour les applications non-serveurs.
ContainsFileDropList	Détermine si le Presse-papiers contient FileDropList. Cette méthode est disponible uniquement pour les applications non-serveurs.
ContainsImage	Détermine si le Presse-papiers contient une image Cette méthode est disponible uniquement pour les applications non-serveurs.
ContainsText	Détermine si le Presse-papiers contient du texte Cette méthode est disponible uniquement pour les applications non-serveurs.
GetAudioStream	Récupère un flux de données audio du Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
GetData	Récupère les données contenues dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
GetDataObject	Récupère un objet de données du Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
GetFileDropList	Récupère un FileDropList du Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
GetImage	Récupère une image du Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
GetText	Récupère le texte contenu dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
SetAudio	Écrit les données audio dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
SetData	Écrit dans le Presse-papiers des données au format personnalisé spécifié. Cette méthode est disponible uniquement pour les applications non-serveurs.

SetDataObject	Écrit un objet de données dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
SetFileDropList	Écrit un FileFropList dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
SetImage	Écrit une image dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.
SetText	Écrit du texte dans le Presse-papiers. Cette méthode est disponible uniquement pour les applications non-serveurs.

Voir aussi

Tâches

Comment : effacer le Presse-papiers en Visual Basic
Comment : lire le Presse-papiers en Visual Basic
Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic
Comment : écrire dans le Presse-papiers en Visual Basic
Comment : récupérer une image du Presse-papiers en Visual Basic
Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.Clear, méthode

Efface le Presse-papiers.

```
' Usage
My.Computer.Clipboard.Clear()
' Declaration
Public Sub Clear()
```

Notes

La classe UIPermission contrôle l'accès au Presse-papiers et la valeur d'énumération UIPermissionClipboard associée indique le niveau d'accès. Pour plus d'informations, consultez Considérations supplémentaires sur la sécurité des Windows Forms.

Tâches

Le tableau suivant répertorie un exemple de tâche impliquant la méthode **My.Computer.Clipboard.Clear**.

Pour	**Voir**
Effacer le Presse-papiers	Comment : effacer le Presse-papiers en Visual Basic

Exemple

Dans cet exemple, le Presse-papiers est nettoyé.

```
My.Computer.Clipboard.Clear()
```

Cela supprime toutes les données du Presse-papiers.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : effacer le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard, membres de l'objet
System.Windows.Forms.Clipboard.Clear

My.Computer.Clipboard.ContainsAudio, méthode

Indique si le Presse-papiers contient des données audio.

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsAudio()
' Declaration
Public Function ContainsAudio() As Boolean
```

Valeur de retour

True si des données audio sont stockées dans le Presse-papiers ; sinon **False**.

Notes
Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.ContainsAudio**.

Pour	Consultez
Déterminez le type de données stocké dans le Presse-papiers	Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Exemple

Cet exemple détermine si le Presse-papiers contient des données audio et affiche le résultat.

VB

```
If My.Computer.Clipboard.ContainsAudio() Then
   MsgBox("The clipboard contains audio data.")
Else
   MsgBox("The clipboard does not contain audio data.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.GetAudioStream, méthode

My.Computer.Clipboard.SetAudio, méthode

System.Windows.Forms.Clipboard.ContainsAudio

Autres ressources

Lecture de sons

My.Computer.Clipboard.ContainsData, méthode

Indique si le Presse-papiers contient des données au format personnalisé spécifié.

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsData(format)
' Declaration
Public Function ContainsData( _
   ByVal format As String _
) As Boolean
```

Paramètres

format

String. Nom du format personnalisé à vérifier. Requis.

Valeur de retour

True si les données au format personnalisé spécifié sont stockées dans le Presse-papiers ; sinon **False**.

Exceptions

Cette méthode ne lève pas d'exceptions.

Notes

Si vous avez placé les données au format personnalisé dans le Presse-papiers, cette méthode vous permet de vérifier que le Presse-papiers contient des données dans ce format.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.ContainsData**.

Pour	Consultez
Déterminez le type de données stocké dans le Presse-papiers	Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Exemple

Cet exemple vérifie s'il existe des données au format personnalisé `specialFormat`.

VB

```
If My.Computer.Clipboard.ContainsData("specialFormat") Then
   MsgBox("Data found.")
End If
```

Remplacez `specialFormat` par le nom du format personnalisé que vous souhaitez vérifier.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**

Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.GetData, méthode
My.Computer.Clipboard.GetDataObject, méthode
My.Computer.Clipboard.SetData, méthode
My.Computer.Clipboard.SetDataObject, méthode
ContainsData

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.ContainsFileDropList, méthode

Retourne un **Boolean** qui indique si le Presse-papiers contient une liste déroulante de fichiers.

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsFileDropList()
' Declaration
Public Function ContainsFileDropList() As Boolean
```

Valeur de retour

True si une liste déroulante de fichiers est stockée dans le Presse-papiers; sinon **False**.

Notes

Une liste déroulante de fichiers est une collection de chaînes contenant des informations relatives au chemin d'accès pour les fichiers.

Exemple

Cet exemple détermine s'il y a une liste déroulante de fichiers sur le Presse-papiers et ajoute la liste au **ListBox** `lstFiles` s'ils existent.

VB

```
If My.Computer.Clipboard.ContainsFileDropList Then
   Dim filelist As System.Collections.Specialized.StringCollection
   filelist = My.Computer.Clipboard.GetFileDropList()
   For Each filePath As String In filelist
       lstFiles.Items.Add(filePath)
   Next
End If
```

Ce code créera une exception s'il n'y a aucun **ListBox** nommé `lstFiles`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.GetFileDropList, méthode

My.Computer.Clipboard.SetFileDropList, méthode

System.Windows.Forms.Clipboard.ContainsFileDropList

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.ContainsImage, méthode

Retourne **Boolean** indiquant si une image est stockée dans le Presse-papiers.

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsImage()
' Declaration
Public Function ContainsImage() As Boolean
```

Valeur de retour

True si une image est stockée dans le Presse-papiers ; sinon **False**.

Exceptions

Cette méthode ne lève pas d'exceptions.

Notes

La méthode nécessite AllClipboard.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.ContainsImage**.

Pour	Consultez
Déterminez si le Presse-papiers contient une image	Comment : récupérer une image du Presse-papiers en Visual Basic

Exemple

Cet exemple vérifie si le Presse-papiers contient une image, l'obtient le cas échéant et l'ajoute à `PictureBox1`.

VB

```
If My.Computer.Clipboard.ContainsImage() Then
   Dim grabpicture As System.Drawing.Image
   grabpicture = My.Computer.Clipboard.GetImage()
   PictureBox1.Image = grabpicture
End If
```

Cet exemple dépend de l'existence de **PictureBox** nommé `PictureBox1`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non

Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.GetImage, méthode

My.Computer.Clipboard.SetImage, méthode

System.Windows.Forms.Clipboard.ContainsImage

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.ContainsText, méthode

Détermine le Presse-papiers contient du texte.

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsText()
Dim value As Boolean = My.Computer.Clipboard.ContainsText(format)
' Declaration
Public Function ContainsText() As Boolean
' -or-
Public Function ContainsText( _
   ByVal format As System.Windows.Forms.TextDataFormat _
) As Boolean
```

Paramètres

format

TextDataFormat. Si spécifié, identifie le format de texte à vérifier. Requis.

Valeur de retour

True si le Presse-papiers contient du texte ; sinon **False**.

Notes

Les formats possibles sont CommaSeparatedValue, Html, Rtf et UnicodeText.

La méthode nécessite AllClipboard.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.ContainsText**.

Pour	**Consultez**
Déterminez si le Presse-papiers contient du texte	Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Exemple

Cet exemple détermine si le texte HTML est stocké dans le Presse-papiers et le lit le cas échéant.

VB

```
If My.Computer.Clipboard.ContainsText _
(System.Windows.Forms.TextDataFormat.Html) Then
   Dim clipText As String = My.Computer.Clipboard.GetText()
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**

Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.GetText, méthode
My.Computer.Clipboard.SetText, méthode
System.Windows.Forms.TextDataFormat
System.Windows.Forms.Clipboard.ContainsText

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.GetAudioStream, méthode

Récupère un flux de données audio du Presse-papiers.

```
' Usage
Dim value As System.IO.Stream = My.Computer.Clipboard.GetAudioStream()
' Declaration
Public Function GetAudioStream() As System.IO.Stream
```

Valeur de retour

Stream

Notes

Exemple

Cet exemple récupère un flux de données audio du Presse-papiers et le lit.

VB

```
If My.Computer.Clipboard.ContainsAudio Then
   Dim song As System.IO.Stream
   song = My.Computer.Clipboard.GetAudioStream
   My.Computer.Audio.Play(song, AudioPlayMode.WaitToComplete)
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.ContainsAudio, méthode
My.Computer.Clipboard.SetAudio, méthode
System.IO.Stream

System.Windows.Forms.Clipboard.GetAudioStream

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.GetData, méthode

Récupère des données dans un format personnalisé du Presse-papiers.

```
' Usage
Dim value As Object = My.Computer.Clipboard.GetData(format)
' Declaration
Public Function GetData( _
   ByVal format As String _
) As Object
```

Paramètres

format

String. Nom du format de données. Requis.

Valeur de retour

Objet.

Notes

Exemple

Cet exemple lit des données dans le `specialformat` de format du Presse-papiers et l'écrit dans un fichier.

VB

```
Dim someData As Object
someData = My.Computer.Clipboard.GetData("specialformat")
My.Computer.FileSystem.WriteAllBytes("C:\\mylogfile", someData, True)
```

Remplacez `specialformat` par le format de données personnalisé que vous souhaitez récupérer.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsData, méthode

My.Computer.Clipboard.GetDataObject, méthode

My.Computer.Clipboard.SetData, méthode

My.Computer.Clipboard.SetDataObject, méthode

GetData

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.GetDataObject, méthode

Récupère des données du Presse-papiers en tant que IDataObject.

```
' Usage
Dim value As System.Windows.Forms.IDataObject = My.Computer.Clipboard.GetDataObject()
' Declaration
Public Function GetDataObject() As System.Windows.Forms.IDataObject
```

Valeur de retour

IDataObject

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Exemple

Cet exemple lit des données du Presse-papiers dans le formulaire d'un **IDataObject** puis l'écrit dans un fichier.

VB

```
Dim someData As Object
someData = My.Computer.Clipboard.GetDataObject()
My.Computer.FileSystem.WriteAllBytes("C:\mylogfile", someData, True)
```

Remplacez `C:\mylogfile` par le nom du fichier dans lequel vous souhaitez écrire.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsData, méthode

My.Computer.Clipboard.GetData, méthode
My.Computer.Clipboard.SetData, méthode
My.Computer.Clipboard.SetDataObject, méthode
System.Windows.Forms.IDataObject
System.Windows.Forms.Clipboard.GetDataObject

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.GetFileDropList, méthode

Récupère du Presse-papiers une collection des chaînes représentant des noms de fichiers.

```
' Usage
Dim value As System.Collections.Specialized.StringCollection = My.Computer.Clipboard.GetFil
eDropList()
' Declaration
Public Function GetFileDropList() As System.Collections.Specialized.StringCollection
```

Valeur de retour

StringCollection

Notes

Cette méthode est disponible uniquement pour les applications non-serveurs.

Une liste déroulante de fichiers est une collection de chaînes contenant des informations relatives au chemin d'accès des fichiers.

Exemple

Cet exemple détermine s'il existe une liste déroulante de fichiers dans le Presse-papiers et ajoute la liste au **ListBox** `lstFiles` s'ils existent.

VB

```
If My.Computer.Clipboard.ContainsFileDropList Then
   Dim filelist As System.Collections.Specialized.StringCollection
   filelist = My.Computer.Clipboard.GetFileDropList()
   For Each filePath As String In filelist
       lstFiles.Items.Add(filePath)
   Next
End If
```

Ce code crée une exception s'il n'existe aucun **ListBox** nommé `lstFiles`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**

Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsFileDropList, méthode

My.Computer.Clipboard.SetFileDropList, méthode

System.Collections.Specialized.StringCollection

System.Windows.Forms.Clipboard.GetFileDropList

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.GetImage, méthode

Récupère une image du Presse-papiers.

```
' Usage
Dim value As System.Drawing.Image = My.Computer.Clipboard.GetImage()
' Declaration
Public Function GetImage() As System.Drawing.Image
```

Valeur de retour

Image

Notes

La méthode **GetImage** retournera **Nothing** s'il n'y a pas de données dans le Presse-papiers qui sont une image.

La méthode nécessite AllClipboard.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.GetImage**.

Pour	Consultez
Récupérer une image du Presse-papiers.	Comment : récupérer une image du Presse-papiers en Visual Basic

Exemple

Cet exemple vérifie s'il y a une image dans le Presse-papiers avant de la récupérer et de l'assigner à `PictureBox1`.

VB

```
If My.Computer.Clipboard.ContainsImage() Then
   Dim grabpicture As System.Drawing.Image
   grabpicture = My.Computer.Clipboard.GetImage()
   PictureBox1.Image = grabpicture
End If
```

Pour que cet exemple fonctionne correctement, il doit exister un **PictureBox** nommé `PictureBox1` sur votre formulaire.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**

Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : déterminer le type de fichier stocké dans le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsImage, méthode

My.Computer.Clipboard.SetImage, méthode

System.Drawing.Image

System.Windows.Forms.Clipboard.GetImage

My.Computer.Clipboard.GetText, méthode

Récupère le texte contenu dans le Presse-papiers.

```
' Usage
Dim value As String = My.Computer.Clipboard.GetText()
Dim value As String = My.Computer.Clipboard.GetText(format)
' Declaration
Public Function GetText() As String
' -or-
Public Function GetText( _
   ByVal format As System.Windows.Forms.TextDataFormat _
) As String
```

Paramètres

format

TextDataFormat. S'il est spécifié, identifie le format de texte qui doit être récupéré. La valeur par défaut est CommaSeparatedValue. Requis.

Valeur de retour

String.

Exceptions

Cette méthode ne lève pas d'exceptions.

Notes

Les formats possibles sont **CommaSeparatedValue**, Html, Rtf et UnicodeText.

La méthode nécessite AllClipboard.

Cette méthode retourne une chaîne vide s'il n'y a pas de données dans le Presse-papiers qui correspond au format spécifié.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.GetText**.

Pour	Consultez
Lire le texte contenu dans le Presse-papiers.	Comment : lire le Presse-papiers en Visual Basic

Exemple

Cet exemple lit le texte du Presse-papiers dans la chaîne `textOnClipboard`.

VB

```
Dim textOnClipboard As String = My.Computer.Clipboard.GetText()
```

Cet exemple ne fonctionne pas s'il n'y a pas de texte dans le Presse-papiers.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**

Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire le Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsText, méthode

My.Computer.Clipboard.SetText, méthode

System.Windows.Forms.TextDataFormat

System.Windows.Forms.Clipboard.GetText

My.Computer.Clipboard.SetAudio, méthode

Écrit les données audio dans le Presse-papiers.

```
' Usage
My.Computer.Clipboard.SetAudio(audioBytes)
My.Computer.Clipboard.SetAudio(audioStream)
' Declaration
Public Sub SetAudio( _
   ByVal audioBytes As Byte() _
)
' -or-
Public Sub SetAudio( _
   ByVal audioStream As System.IO.Stream _
)
```

Paramètres

audioBytes

Tableau de **Byte**. Données audio à écrire dans le Presse-papiers. Requis.

audioStream

Stream Données audio à écrire dans le Presse-papiers. Requis.

Exceptions

Aucune exception n'est levée par cette méthode.

Notes

Cette méthode peut être utilisée pour écrire les données audio comme un tableau d'octets ou un flux de données audio.

Remarque de sécurité
Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.SetAudio**.

Pour	**Consultez**
Écrire des données audio dans le Presse-papiers.	Comment : enregistrer un flux audio dans le Presse-papiers en Visual Basic

Exemple

Cet exemple crée le tableau d'octets `musicReader`, lit le fichier `cool.wav`, puis l'écrit dans le Presse-papiers.

VB

```
Dim musicReader As Byte()
musicReader = My.Computer.FileSystem.ReadAllBytes("cool.wav")
My.Computer.Clipboard.SetAudio(musicReader)
```

Remplacez `cool.wav` par le nom et le chemin d'accès du fichier que vous souhaitez lire.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.ContainsAudio, méthode
My.Computer.Clipboard.GetAudioStream, méthode
System.IO.Stream
System.Windows.Forms.Clipboard.SetAudio

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.SetData, méthode

Écrit des données dans un format personnalisé dans le Presse-papiers.

```
' Usage
My.Computer.Clipboard.SetData(format ,data)
' Declaration
Public Sub SetData( _
   ByVal format As String, _
   ByVal data As Object _
)
```

Paramètres

format

String. Format de données. Requis.

data

Object. Objet de données à écrire dans le Presse-papiers. Requis.

Exceptions

Cette méthode ne lève pas d'exceptions.

Notes

Remarque de sécurité
Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Exemple

Cet exemple écrit le `dataChunk` du **DataObject** dans le Presse-papiers au format personnalisé `specialFormat`.

VB

```
My.Computer.Clipboard.SetData("specialFormat", dataChunk)
```

Cet exemple dépend de l'existence du format de données personnalisé `specialFormat`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non

Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.ContainsData, méthode
My.Computer.Clipboard.GetData, méthode
My.Computer.Clipboard.GetDataObject, méthode
My.Computer.Clipboard.SetDataObject, méthode
SetData

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.SetDataObject, méthode

Écrit un DataObject dans le Presse-papiers.

```
' Usage
My.Computer.Clipboard.SetDataObject(data)
' Declaration
Public Sub SetDataObject( _
   ByVal data As System.Windows.Forms.DataObject _
)
```

Paramètres

data

DataObject. Objet de données à écrire dans le Presse-papiers. Requis.

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Remarque de sécurité
Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Exemple

Cet exemple écrit l'objet de données `dataChunk` dans le Presse-papiers.

VB

```
My.Computer.Clipboard.SetDataObject(dataChunk)
```

Cet exemple dépend de l'existence de l'objet de données `dataChunk`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsData, méthode

My.Computer.Clipboard.GetData, méthode

My.Computer.Clipboard.GetDataObject, méthode

My.Computer.Clipboard.SetData, méthode

SetDataObject

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.SetFileDropList, méthode

Écrit une collection de chaînes représentant des chemins d'accès vers le Presse-papiers.

```
' Usage
My.Computer.Clipboard.SetFileDropList(filePaths)
' Declaration
Public Sub SetFileDropList( _
   ByVal filePaths As System.Collections.Specialized.StringCollection _
)
```

Paramètres

filePaths

StringCollection. Liste des noms de fichiers. Requis.

Notes

Une liste déroulante de fichiers est une collection de chaînes représentant des noms de fichiers.

Remarque de sécurité
Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Exemple

Cet exemple obtient la collection de noms de fichiers dans **MyDocuments**, le convertit en une liste déroulante de fichiers et l'écrit dans le Presse-papiers.

VB

```
Dim list As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
list = My.Computer.FileSystem.GetFiles _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
Dim listReader As New System.Collections.Specialized.StringCollection
For Each item As String In list
   listReader.Add(item)
Next
My.Computer.Clipboard.SetFileDropList(listReader)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non

Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.ContainsFileDropList, méthode
My.Computer.Clipboard.GetFileDropList, méthode
System.Collections.Specialized.StringCollection
SetFileDropList

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.SetImage, méthode

Écrit une image dans le Presse-papiers.

```
' Usage
My.Computer.Clipboard.SetImage(image)
' Declaration
Public Sub SetImage( _
   ByVal image As System.Drawing.Image _
)
```

Paramètres

image

Image. Image à écrire. Requis.

Notes

Remarque de sécurité

Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Exemple

Cet exemple écrit l'image `coolPicture` dans le Presse-papiers.

VB

```
My.Computer.Clipboard.SetImage(coolPicture)
```

Cet exemple dépend de l'existence de l'image `coolPicture`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : récupérer une image du Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet

My.Computer.Clipboard.ContainsImage, méthode

My.Computer.Clipboard.GetImage, méthode

System.Drawing.Image

System.Windows.Forms.Clipboard.SetImage(System.Drawing.Image)

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clipboard.SetText, méthode

Écrit du texte dans le Presse-papiers.

```
' Usage
My.Computer.Clipboard.SetText(text)
My.Computer.Clipboard.SetText(text ,format)
' Declaration
Public Sub SetText( _
   ByVal text As String _
)
' -or-
Public Sub SetText( _
   ByVal text As String, _
   ByVal format As System.Windows.Forms.TextDataFormat _
)
```

Paramètres

text

String. Texte à écrire. Requis.

format

TextDataFormat. Format à utiliser lors de l'écriture du texte. La valeur par défaut est UnicodeText. Requis.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- *text* est une chaîne vide (ArgumentException).
- *text* a la valeur **Nothing** (ArgumentNullException).

Notes

Les formats possibles sont CommaSeparatedValue, Html, Rtf et **UnicodeText**.

La méthode nécessite AllClipboard.

Les formats de Presse-papiers antérieurs ne sont pas conservés.

Remarque de sécurité
Étant donné que d'autres utilisateurs peuvent accéder au Presse-papiers, ne l'utilisez pas pour stocker des informations sensibles, telles que des mots de passe ou des données confidentielles.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Clipboard.SetText**.

Pour	Consultez
Écrire du texte dans le Presse-papiers	Comment : écrire dans le Presse-papiers en Visual Basic

Exemple

Cet exemple écrit la chaîne `This is a test string.` dans le Presse-papiers.

VB

```
My.Computer.Clipboard.SetText("This is a test string.")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : ClipboardProxy (fournit l'accès à Clipboard)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clipboard, objet
My.Computer.Clipboard.ContainsText, méthode
My.Computer.Clipboard.GetText, méthode
System.Windows.Forms.TextDataFormat
System.Windows.Forms.Clipboard.SetText

Autres ressources

Stockage de données dans le Presse-papiers et lecture du Presse-papiers

My.Computer.Clock, objet

Fournit des propriétés pour accéder à l'heure locale actuelle et au temps UTC (Universal Coordinated Time), équivalent à l'heure GMT (Greenwich Mean Time), à partir de l'horloge système.

Notes

L'objet **My.Computer.Clock** fournit des propriétés pour rechercher l'heure locale actuelle de l'ordinateur, ainsi que l'heure UTC. Il expose également le nombre de millisecondes de l'horloge système de l'ordinateur.

Exemple

Cet exemple utilise la propriété **My.Computer.Clock.LocalTime** pour afficher l'heure locale de l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Current local time: " & My.Computer.Clock.LocalTime)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Clock

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de l'objet My.Computer.Clock
My.Computer, objet
Microsoft.VisualBasic.Devices.Clock

Membres de l'objet My.Computer.Clock

L'objet My.Computer.Clock fournit des propriétés pour accéder à l'heure locale actuelle et au temps UTC (Universal Coordinated Time), équivalent à l'heure GMT (Greenwich Mean Time), à partir de l'horloge système.

Propriétés

Propriété	**Description**
GmtTime	Obtient un objet **Date** qui contient la date et l'heure locales actuelles sur cet ordinateur, exprimées en heure UTC (GMT).
LocalTime	Obtient un objet **Date** qui contient la date et l'heure locales actuelles sur cet ordinateur.
TickCount	Obtient le nombre de millisecondes de l'horloge système de l'ordinateur.

Voir aussi

Référence

My.Computer.Clock, objet

My.Computer.Clock.GmtTime, propriété

Obtient un objet **Date** qui contient la date et l'heure locales actuelles sur l'ordinateur, exprimées en heure UTC (GMT).

```
' Usage
Dim value As Date = My.Computer.Clock.GmtTime
' Declaration
Public ReadOnly Property GmtTime As Date
```

Valeur de retour

Objet **Date** qui contient la date et l'heure actuelles exprimées en heure UTC (GMT).

Notes

La propriété **My.Computer.Clock.GmtTime** a le même comportement que la propriété System.DateTime.UtcNow.

Exemple

Cet exemple utilise la propriété **My.Computer.Clock.GmtTime** pour afficher l'heure GMT de l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Current GMT time: " & My.Computer.Clock.GmtTime)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Clock

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clock, objet
My.Computer.Clock.LocalTime, propriété
System.DateTime.UtcNow
Microsoft.VisualBasic.Devices.Clock.GmtTime

My.Computer.Clock.LocalTime, propriété

Obtient un objet **Date** qui contient la date et l'heure locales actuelles sur cet ordinateur.

```
' Usage
Dim value As Date = My.Computer.Clock.LocalTime
' Declaration
Public ReadOnly Property LocalTime As Date
```

Valeur de retour

Objet **Date** qui contient la date et l'heure locales actuelles.

Notes

La propriété **My.Computer.Clock.LocalTime** a le même comportement que la propriété System.DateTime.Now.

Exemple

Cet exemple utilise la propriété **My.Computer.Clock.LocalTime** pour afficher l'heure locale de l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Current local time: " & My.Computer.Clock.LocalTime)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Clock

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clock, objet
My.Computer.Clock.GmtTime, propriété
System.DateTime.Now
Microsoft.VisualBasic.Devices.Clock.LocalTime

My.Computer.Clock.TickCount, propriété

Obtient le nombre de millisecondes de l'horloge système de l'ordinateur.

```
' Usage
Dim value As Integer = My.Computer.Clock.TickCount
' Declaration
Public ReadOnly Property TickCount As Integer
```

Valeur de retour

Integer qui contient le nombre de millisecondes de l'horloge système de l'ordinateur.

Notes

La propriété **TickCount** fournit un accès à l'horloge système de l'ordinateur qui fonctionne lorsque l'ordinateur est actif. La résolution de minuterie n'est pas inférieure à 500 millisecondes.

Vous pouvez utiliser cette propriété pour définir le comportement de votre application en fonction de sa durée de fonctionnement, ou vous pouvez l'utiliser pour étiqueter des événements ; ces deux utilisations sont indépendantes de l'horloge de l'ordinateur.

Attention

Lorsque la valeur de la propriété **TickCount** atteint la valeur entière maximale (MaxValue), elle passe à la valeur entière mini male (MinValue), un nombre négatif, puis continue à incrémenter.

Si l'ordinateur fonctionne en continu, **TickCount** incrémente de zéro à la valeur entière maximale dans environ 24,9 jours.

La propriété **TickCount** incrémente uniquement lorsque le système d'exploitation est en cours d'exécution ; elle s'arrête lorsque l'ordinateur active certains modes d'économie d'énergie, par exemple la mise en veille ou la mise en veille prolongée. La propriété **TickCount** n'est pas liée au paramètre de l'horloge de l'ordinateur.

Utilisez la propriété My.Computer.Clock.LocalTime ou My.Computer.Clock.GmtTime pour obtenir la date et l'heure locales actuelles sur cet ordinateur.

La propriété **My.Computer.Clock.TickCount** a le même comportement que la propriété System.Environment.TickCount.

Exemple

L'exemple suivant utilise la propriété **My.Computer.Clock.TickCount** pour exécuter une tâche dans une boucle pendant un nombre de secondes spécifique, même si l'horloge système de l'ordinateur change pendant son exécution.

VB

```
Public Sub LoopTask(ByVal secondsToRun As Integer)
    Dim startTicks As Integer = My.Computer.Clock.TickCount
    Do While IsTimeUp(startTicks, secondsToRun)
        ' Code to run for at least secondsToRun seconds goes here.
    Loop
End Sub

Private Function IsTimeUp( _
    ByVal startTicks As Integer, _
    ByVal seconds As Integer _
) As Boolean
    ' This function throws an overflow exception if the
    ' tick count difference is greater than 2,147,483,647,
    ' about 24 days for My.Computer.Clock.TickCount.

    ' Use UInteger to simplify the code for roll over.
    Dim uStart As UInteger = _
        CUInt(CLng(startTicks) - Integer.MinValue)
    Dim uCurrent As UInteger = _
        CUInt(CLng(My.Computer.Clock.TickCount) - Integer.MinValue)
```

```
        ' Calculate the tick count difference.
        Dim tickCountDifference As UInteger
        If uStart <= uCurrent Then
            tickCountDifference = uCurrent - uStart
        Else
            ' Tick count rolled over.
            tickCountDifference = UInteger.MaxValue - (uStart - uCurrent)
        End If

        ' Convert seconds to milliseconds and compare.
        Return CInt(tickCountDifference) < (seconds * 1000)
    End Function
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Clock

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Clock, objet
System.Environment.TickCount
Microsoft.VisualBasic.Devices.Clock.TickCount

My.Computer.FileSystem, objet

Fournit des propriétés et des méthodes pour utiliser les lecteurs, les fichiers et les répertoires.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.FileSystem**.

Pour	**Consultez**
Lire un fichier texte	Comment : lire des fichiers texte dans Visual Basic
Lire un fichier texte délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier texte à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Lire un fichier texte avec plusieurs formats	Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic
Lire un fichier binaire	Comment : lire des fichiers binaires dans Visual Basic
Lire des fichiers texte contenus dans le répertoire **MyDocuments**	Comment : lire des fichiers texte existants dans Mes documents (Visual Basic)
Lire un fichier texte avec StreamReader	Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)
Écrire dans un fichier texte	Comment : insérer du texte dans des fichiers dans Visual Basic
Effectuer un ajout dans un fichier texte	Comment : effectuer un ajout à des fichiers texte dans Visual Basic
Écrire dans un fichier binaire	Comment : écrire dans des fichiers binaires dans Visual Basic
Écrire dans des fichiers texte contenus dans le répertoire **MyDocuments**	Comment : insérer du texte dans les fichiers du répertoire Mes Documents dans Visual Basic
Écrire dans un fichier texte avec **StreamWriter**	Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic
Copier des fichiers avec un modèle spécifique	Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic
Copier un fichier vers le même répertoire	Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic
Copier un fichier vers un autre répertoire	Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic
Créer un fichier	Comment : créer un fichier dans Visual Basic
Supprimer un fichier	Comment : supprimer un fichier dans Visual Basic

Supprimer tous les fichiers d'un répertoire	Comment : supprimer tous les fichiers d'un répertoire dans Visual Basic
Rechercher des fichiers avec un modèle spécifique	Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic
Déplacer un fichier	Comment : déplacer un fichier dans Visual Basic
Déplacer une collection de fichiers	Comment : déplacer une collection de fichiers dans Visual Basic
Renommer un fichier	Comment : renommer un fichier dans Visual Basic
Renommer un répertoire	Comment : renommer un répertoire dans Visual Basic
Copier un répertoire dans un autre répertoire	Comment : copier un répertoire dans un autre répertoire dans Visual Basic
Créer un répertoire	Comment : créer un répertoire dans Visual Basic
Supprimer un répertoire	Comment : supprimer un répertoire dans Visual Basic
Rechercher des sous-répertoires avec un modèle spécifique	Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic
Obtenir la collection de fichiers d'un répertoire	Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Déterminer le nombre de fichiers contenus dans un répertoire	Comment : obtenir le nombre de fichiers d'un répertoire dans Visual Basic
Déplacer un répertoire	Comment : déplacer un répertoire dans Visual Basic
Déplacer le contenu d'un répertoire	Comment : déplacer le contenu d'un répertoire dans Visual Basic
Lire le répertoire **MyDocuments**	Comment : récupérer le contenu du répertoire Mes Documents dans Visual Basic
Analyser un chemin d'accès	Comment : analyser des chemins d'accès dans Visual Basic

Exemple

Cet exemple détermine si le dossier `C:\backup\logs` existe et vérifie ses propriétés.

VB

```
Dim logInfo As System.IO.DirectoryInfo
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    logInfo = My.Computer.FileSystem.GetDirectoryInfo _
    ("C:\backup\logs")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

My.Computer.FileSystem, membres de l'objet
My.Computer.FileSystem.SpecialDirectories, objet
My.Computer, objet
Microsoft.VisualBasic.FileIO.FileSystem

My.Computer.FileSystem, membres de l'objet

L'objet My.Computer.FileSystem fournit des propriétés et des méthodes pour utiliser les lecteurs, les fichiers et les répertoires.

Propriétés

Propriété	Description
CurrentDirectory	Obtient le répertoire actif.
Lecteurs	Obtient des informations sur les lecteurs.
SpecialDirectories	Retourne l'objet My.Computer.FileSystem.SpecialDirectories qui peut être utilisé pour accéder à des répertoires spéciaux, par exemple **Temp** ou **MyDocuments**.

Méthodes

Méthode	Description
CombinePath	Retourne un chemin d'accès combiné correctement mis en forme en tant que **String**.
CopyDirectory	Copie un répertoire.
CopyFile	Copie un fichier.
CreateDirectory	Crée un répertoire.
DeleteDirectory	Supprime un répertoire.
DeleteFile	Supprime un fichier.
DirectoryExists	Retourne **Boolean** qui indique si un répertoire existe.
FileExists	Retourne **Boolean** qui indique si un fichier existe.
My.Computer.FileSystem.FindInFiles, méthode	Retourne une collection en lecture seule de chaînes qui représentent le nom des fichiers contenant le texte spécifié.
GetDirectories	Retourne une collection **String** représentant les chemins d'accès de sous-répertoires présents dans un répertoire défini.
GetDirectoryInfo	Retourne un objet DirectoryInfo pour le chemin d'accès spécifié.
GetDriveInfo	Retourne un objet DriveInfo pour le chemin d'accès spécifié.
GetFileInfo	Retourne un objet FileInfo pour le chemin d'accès spécifié.
GetFiles	Retourne une collection **String** en lecture seule qui représente le nom des fichiers d'un répertoire.
GetParentPath	Retourne **String** qui représente le chemin d'accès absolu du parent du chemin d'accès fourni.
MoveDirectory	Déplace un répertoire.
MoveFile	Déplace un fichier.

OpenTextFieldParser	Ouvre TextFieldParser.
OpenTextFileReader	Ouvre TextReader.
OpenTextFileWriter	Ouvre TextWriter.
ReadAllBytes	Lit un fichier binaire.
ReadAllText	Lit un fichier texte.
RenameDirectory	Renomme un répertoire.
RenameFile	Renomme un fichier.
WriteAllBytes	Écrit dans un fichier binaire.
WriteAllText	Écrit dans un fichier texte.

Voir aussi

Tâches

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.CombinePath, méthode

Associe deux chemins d'accès et retourne un chemin d'accès correctement mis en forme.

```
' Usage
Dim value As String = My.Computer.FileSystem.CombinePath(baseDirectory ,relativePath)
' Declaration
Public Function CombinePath( _
   ByVal baseDirectory As String, _
   ByVal relativePath As String _
) As String
```

Paramètres

baseDirectory

String. Premier chemin d'accès à associer. Obligatoire.

relativePath

String. Deuxième chemin d'accès à associer. Obligatoire.

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est incorrect (ArgumentException).

Notes

Cette méthode supprime les caractères de marque de barre oblique excédentaires pour créer un chemin d'accès correctement mis en forme.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.CombinePath**.

Pour	Consultez
Associer un chemin d'accès de répertoire et un nom de fichier	Comment : analyser des chemins d'accès dans Visual Basic

Exemple

Cet exemple associe un chemin d'accès de répertoire et un nom de fichier pour créer un chemin d'accès correctement mis en forme.

VB

```
Dim fullPath As String
fullPath = My.Computer.FileSystem.CombinePath _
("C:\Documents and Settings\All Users\Documents\My Pictures", "picture.jpg")
```

Cet exemple associe deux chemins d'accès pour créer un chemin d'accès correctement mis en forme.

VB

```
Dim fullPath As String
fullPath = My.Computer.FileSystem.CombinePath _
("C:\Dir1\Dir2\Dir3", "..\Dir4\Dir5\File.txt")
```

Cet exemple retourne `C:\Dir1\Dir2\Dir4\Dir5\File.txt`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.GetParentPath, méthode
CombinePath

My.Computer.FileSystem.CopyDirectory, méthode

Copie un répertoire dans un autre répertoire.

```
' Usage
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName)
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName ,overwri
te)
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI)
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI
,onUserCancel)
' Declaration
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String _
)
' -or-
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

sourceDirectoryName

String. Répertoire à copier. Obligatoire.

destinationDirectoryName

String. Emplacement vers lequel le répertoire doit être copié. Obligatoire.

overwrite

Boolean. Si des fichiers existants doivent être remplacés. La valeur par défaut est **False**. Obligatoire.

showUI

UIOption. Si besoin d'un suivi visuel de l'avancement de l'opération. La valeur par défaut est **UIOption.OnlyErrorDialogs**. Obligatoire.

onUserCancel

UICancelOption. Spécifie l'action à effectuer si l'utilisateur clique sur **Annuler** pendant l'opération. La valeur par défaut est ThrowException. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le nouveau nom spécifié pour le répertoire contient le signe deux-points (:) ou une barre oblique (\ ou /) (ArgumentException).

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *destinationDirectoryName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le répertoire source n'existe pas (DirectoryNotFoundException).
- Le répertoire source est un répertoire racine (IOException).
- Le chemin d'accès combiné pointe vers un fichier existant (**IOException**).
- Les chemins d'accès source et cible sont identiques (**IOException**).
- *ShowUI* a la valeur **UIOption.AllDialogs** et l'utilisateur annule l'opération, ou un ou plusieurs fichiers dans le répertoire ne peuvent pas être copiés (OperationCanceledException).
- L'opération est cyclique (InvalidOperationException).
- Le chemin d'accès contient le signe deux-points (:) (NotSupportedException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (**NotSupportedException**).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- Un fichier de destination existe, mais n'est pas accessible (UnauthorizedAccessException).

Notes

Cette méthode copie le contenu du répertoire, ainsi que le répertoire lui-même. Si le répertoire cible n'existe pas, il est créé. Si un répertoire portant le même nom existe à l'emplacement cible, le contenu des deux répertoires est fusionné. Vous pouvez spécifier un nouveau nom pour le répertoire pendant l'opération.

Lors de la copie de fichiers dans un répertoire, des exceptions provoquées par un fichier spécifique, comme un fichier existant au cours d'une fusion alors que *overwrite* a la valeur **False**, peuvent être levées. Lorsque de telles exceptions sont levées, elles sont consolidées en une seule exception dont la propriété `Data` comporte des entrées sous forme de IDictionary où le chemin d'accès au fichier ou au répertoire constitue la clé et le message d'exception spécifique est contenu dans la valeur correspondante. Utilisez **For...Each** pour énumérer les entrées.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.CopyDirectory**.

Pour	**Consultez**
Copier un répertoire	Comment : copier un répertoire dans un autre répertoire dans Visual Basic

Exemple

L'exemple suivant copie le répertoire `TestDirectory1` dans `TestDirectory2`, en remplaçant les fichiers existants.

VB

```
My.Computer.FileSystem.CopyDirectory("C:\TestDirectory1", "C:\TestDirectory2", True)
```

Remplacez `C:\TestDirectory1` et `C:\TestDirectory2` par le chemin d'accès et le nom du répertoire que vous souhaitez copier et l'emplacement vers lequel vous souhaitez le copier.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : SafeSubWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Comment : déplacer un répertoire dans Visual Basic
Comment : déplacer le contenu d'un répertoire dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic
Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic
Comment : déterminer si un répertoire existe dans Visual Basic

Référence

My.Computer.FileSystem, objet
UICancelOption, énumération

My.Computer.FileSystem.CopyFile, méthode

Copie un fichier vers un nouvel emplacement.

```
' Usage
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName)
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName ,overwrite)
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName ,showUI)
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName ,showUI ,onUserCancel)
' Declaration
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

sourceFileName

String. Fichier à copier. Obligatoire.

destinationFileName

String. Emplacement vers lequel le fichier doit être copié. Obligatoire.

overwrite

Boolean. Si les fichiers existants doivent être remplacés. La valeur par défaut est **False**. Obligatoire.

showUI

UIOption. Si besoin d'un suivi visuel de l'avancement de l'opération. La valeur par défaut est **UIOption.OnlyErrorDialogs**. Obligatoire.

onUserCancel

UICancelOption. Spécifie l'action à effectuer si l'utilisateur clique sur **Annuler** pendant l'opération. La valeur par défaut est ThrowException. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le système n'a pas pu récupérer le chemin d'accès absolu (**ArgumentException**).

- *destinationFileName* contient des informations relatives au chemin d'accès (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *destinationFileName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Le chemin d'accès combiné pointe vers un répertoire existant (IOException).
- Le fichier de destination existe et *overwrite* a la valeur **False** (**IOException**).
- L'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (**IOException**).
- Un fichier du répertoire cible qui porte le même nom est en cours d'utilisation (**IOException**).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- *UICancelOption* a la valeur **ThrowException**, et l'utilisateur a annulé l'opération (OperationCanceledException).
- *UICancelOption* a la valeur **ThrowException**, et une erreur d'E/S non spécifiée s'est produite (**OperationCanceledException**).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Notes

CopyFile ne conserve pas les entrées du contrôle d'accès (ACE, Access Control Entries). Le fichier nouvellement créé hérite des entrées ACE par défaut du répertoire dans lequel il est créé.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.CopyFile**.

Pour	Consultez
Copier un fichier vers le même répertoire.	Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic
Copier un fichier vers un autre répertoire.	Comment : créer une copie d'un fichier dans un autre répertoire dans Visual Basic

Exemple

Cet exemple copie le fichier `Test.txt` dans le répertoire `TestFiles2` sans remplacer les fichiers existants.

VB

```
My.Computer.FileSystem.CopyFile _
("C:\UserFiles\TestFiles\test.txt", _
"C:\UserFiles\TestFiles2")
```

Remplacez les chemins d'accès par ceux que vous souhaitez utiliser dans votre code.

Cet exemple copie le fichier `Test.txt` dans le répertoire `TestFiles2`et le renomme `NewFile.txt`.

VB

```
My.Computer.FileSystem.CopyFile _
("C:\UserFiles\TestFiles\test.txt", _
"C:\UserFiles\TestFiles2", "NewFile.txt", FileIO.UICancelOption.DoNothing)
```

Remplacez les chemins d'accès par ceux que vous souhaitez utiliser dans votre code.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
EnvironmentPermission	Contrôle la possibilité d'accéder à toutes les variables d'environnement. Énumération associée : Unrestricted.
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : **Unrestricted**.
RegistryPermission	Contrôle la possibilité d'accéder aux variables du Registre. Énumération associée : **Unrestricted**.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : SafeSubWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic
Comment : créer une copie d'un fichier dans le même répertoire dans Visual Basic
Comment : copier un répertoire dans un autre répertoire dans Visual Basic
Comment : renommer un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet
UICancelOption, énumération

My.Computer.FileSystem.CreateDirectory, méthode

Crée un répertoire.

```
' Usage
My.Computer.FileSystem.CreateDirectory(directory)
' Declaration
Public Sub CreateDirectory( _
   ByVal directory As String _
)
```

Paramètres

directory

String. Nom et emplacement du répertoire. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le nom du répertoire est incorrect. Il contient par exemple des caractères non valides ou se compose uniquement d'un espace blanc (ArgumentException, classe).
- Le répertoire parent du répertoire à créer est en lecture seule (IOException, classe).
- Le nom du répertoire a la valeur **Nothing** (ArgumentNullException, classe).
- Le nom du répertoire est trop long (PathTooLongException, classe).
- Le nom du répertoire se compose uniquement du signe deux-points (:). (NotSupportedException, classe).
- L'utilisateur n'a pas l'autorisation de créer le répertoire (UnauthorizedAccessException).
- Il existe une situation d'un niveau de confiance partiel dans laquelle l'utilisateur ne dispose pas des autorisations nécessaires (SecurityException).

Notes

Si le répertoire existe déjà, aucune exception n'est levée.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.CreateDirectory**.

Pour	Consultez
Créer un répertoire	Comment : créer un répertoire dans Visual Basic

Exemple

Cet exemple crée le répertoire, `NewDirectory`, dans `C:\Documents and Settings\All Users\Documents`.

VB

```
My.Computer.FileSystem.CreateDirectory _
("C:\Documents and Settings\All Users\Documents\NewDirectory")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer si un répertoire existe dans Visual Basic

Référence

My.Computer.FileSystem, objet

CreateDirectory

My.Computer.FileSystem.CurrentDirectory, propriété

Obtient ou définit le répertoire actif.

```
' Usage
Dim value As String = My.Computer.FileSystem.CurrentDirectory
' Declaration
Public Property CurrentDirectory As String
```

Valeur de retour

String.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide (DirectoryNotFoundException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Notes

CurrentDirectory est une variable d'environnement à l'échelle du système.

Exemple

Cet exemple retourne le répertoire actif et l'affiche dans un message.

VB

```
MsgBox(My.Computer.FileSystem.CurrentDirectory)
```

Cet exemple affecte `C:\TestDirectory` au répertoire actif.

VB

```
My.Computer.FileSystem.CurrentDirectory = "C:\TestDirectory"
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**

Site Web	**Oui**

Autorisations

Des autorisations peuvent ne pas être nécessaires pour définir **CurrentDirectory**.

L'autorisation suivante peut être nécessaire pour lire **CurrentDirectory**:

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

Microsoft.VisualBasic.FileIO.FileSystem.CurrentDirectory

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.DeleteDirectory, méthode

Supprime un répertoire.

```
' Usage
My.Computer.FileSystem.DeleteDirectory(directory ,onDirectoryNotEmpty)
My.Computer.FileSystem.DeleteDirectory(directory ,showUI ,recycle)
My.Computer.FileSystem.DeleteDirectory(directory ,showUI ,recycle ,onUserCancel)
' Declaration
Public Sub DeleteDirectory( _
   ByVal directory As String, _
   ByVal onDirectoryNotEmpty As DeleteDirectoryOption _
)
' -or-
Public Sub DeleteDirectory( _
   ByVal directory As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption _
)
' -or-
Public Sub DeleteDirectory( _
   ByVal directory As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

directory

String. Répertoire à supprimer. Obligatoire.

onDirectoryNotEmpty

Énumération DeleteDirectoryOption. Spécifie l'opération à effectuer lorsqu'un répertoire à supprimer contient des fichiers ou des répertoires. La valeur par défaut est **DeleteDirectoryOption.DeleteAllContents**.

showUI

Énumération UIOption. Spécifie s'il faut effectuer un suivi visuel de l'avancement de l'opération. La valeur par défaut est **UIOption.OnlyErrorDialogs**. Obligatoire.

recycle

Énumération RecycleOption. Spécifie si le fichier supprimé doit être ou non envoyé à la **Corbeille**. La valeur par défaut est **RecycleOption.DeletePermanently**.

onUserCancel

Énumération UICancelOption. Spécifie s'il convient de lever une exception si l'utilisateur clique sur **Annuler**. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès est une chaîne de longueur nulle, est incorrect, se compose uniquement d'un espace blanc ou contient des caractères incorrects (y compris des caractères génériques) (ArgumentException).
- Le chemin d'accès est un chemin de périphérique (commence par \\. \) (**ArgumentException**).
- Le chemin d'accès est **Nothing** (ArgumentNullException).
- Le répertoire n'existe pas ou est un fichier (DirectoryNotFoundException).
- Le répertoire n'est pas vide, et *onDirectoryNotEmpty* a la valeur **ThrowIfDirectoryNonEmpty** (IOException).
- L'utilisateur n'a pas l'autorisation de supprimer le répertoire ou le sous-répertoire (**IOException**).

- Un fichier du répertoire ou du sous-répertoire est en cours d'utilisation (**IOException**).
- Un nom de fichier ou de répertoire contient le signe deux-points (:) (NotSupportedException).
- L'utilisateur annule l'opération ou le répertoire ne peut pas être supprimé (OperationCanceledException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- L'utilisateur n'a pas les autorisations requises (SecurityException).

Notes

Les paramètres *showUI*, *recycle* et *onUserCancel* ne sont pas pris en charge dans les applications qui ne sont pas en mode utilisateur intéractif, telles que les services Windows.

Tâches

Le tableau suivant répertorie un exemple de tâche impliquant la méthode **My.Computer.FileSystem.DeleteDirectory**.

Pour	**Reportez-vous à**
Supprimer un répertoire	Comment : supprimer un répertoire dans Visual Basic

Exemple

L'exemple supprime le répertoire `OldDirectory` uniquement s'il est vide.

VB

```
My.Computer.FileSystem.DeleteDirectory _
("C:\OldDirectory", FileIO.DeleteDirectoryOption.ThrowIfDirectoryNonEmpty)
```

Cet exemple supprime le répertoire `OldDirectory` et tout son contenu.

VB

```
My.Computer.FileSystem.DeleteDirectory _
("C:\OldDirectory", FileIO.DeleteDirectoryOption.DeleteAllContents)
```

Cet exemple supprime le répertoire `OldDirectory` et tout son contenu et demande à l'utilisateur de confirmer la suppression, mais n'envoie pas le contenu à la **Corbeille**.

VB

```
My.Computer.FileSystem.DeleteDirectory _
("C:\OldDirectory", FileIO.UIOption.AllDialogs, FileIO.RecycleOption.DeletePermanently, Fil
eIO.UICancelOption.ThrowException)
```

Cet exemple supprime le répertoire `OldDirectory` et tout son contenu et les envoie à la **Corbeille**, mais n'affiche pas l'avancement de l'opération.

VB

```
My.Computer.FileSystem.DeleteDirectory("C:\OldDirectory", _
 FileIO.UIOption.AllDialogs, FileIO.RecycleOption.SendToRecycleBin, FileIO.UICancelOption.T
hrowException)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au Presse-papiers. Énumération associée : SafeSubWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : supprimer un fichier dans Visual Basic
Comment : supprimer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem, objet
UICancelOption, énumération
RecycleOption, énumération
DeleteDirectoryOption, énumération
UICancelOption, énumération
UIOption, énumération
Microsoft.VisualBasic.FileIO.FileSystem.DeleteDirectory
UICancelOption

My.Computer.FileSystem.DeleteFile, méthode

Supprime un fichier.

```
' Usage
My.Computer.FileSystem.DeleteFile(file)
My.Computer.FileSystem.DeleteFile(file ,showUI ,recycle)
My.Computer.FileSystem.DeleteFile(file ,showUI ,recycle ,onUserCancel)
' Declaration
Public Sub DeleteFile( _
   ByVal file As String _
)
' -or-
Public Sub DeleteFile( _
   ByVal file As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption _
)
' -or-
Public Sub DeleteFile( _
   ByVal file As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

file

String. Nom et chemin d'accès du fichier à supprimer. Obligatoire.

showUI

UIOption. Si besoin d'un suivi visuel de l'avancement de l'opération. La valeur par défaut est **UIOption.OnlyErrorDialogs**. Obligatoire.

recycle

RecycleOption. Spécifie si le fichier supprimé doit être envoyé à la **Corbeille**. La valeur par défaut est **RecycleOption.DeletePermanently**. Obligatoire.

onUserCancel

UICancelOption. Spécifie si une exception est levée lorsque l'utilisateur annule l'opération. La valeur par défaut est **UICancelOption.ThrowException**. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ; il contient une barre oblique finale où un fichier doit être spécifié ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- Le fichier est en cours d'utilisation (IOException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- Le fichier n'existe pas (FileNotFoundException).

- L'utilisateur n'a pas l'autorisation pour supprimer le fichier ou le fichier est en lecture seule (UnauthorizedAccessException).
- Situation d'un niveau de confiance partiel dans laquelle l'utilisateur ne dispose pas des autorisations nécessaires (**SecurityException**).
- L'utilisateur a annulé l'opération et *onUserCancel* a la valeur Microsoft.VisualBasic.FileIO.UICancelOption.ThrowException (OperationCanceledException).

Notes

Les paramètres *showUI*, *recycle* et *onUserCancel* ne sont pas pris en charge dans les applications qui ne sont pas interactives avec l'utilisateur, telles que les services Windows.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.DeleteFile**.

Pour	**Consultez**
Supprimer un fichier	Comment : supprimer un fichier dans Visual Basic
Supprimer tous les fichiers d'un répertoire	Comment : supprimer tous les fichiers d'un répertoire dans Visual Basic

Exemple

Cet exemple supprime le fichier `Test.txt`.

VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt")
```

Cet exemple supprime le fichier `Test.txt` et permet à l'utilisateur de confirmer sa suppression.

VB

```
My.Computer.FileSystem.DeleteFile _
("C:\test.txt", FileIO.UIOption.AllDialogs, FileIO.RecycleOption.SendToRecycleBin, FileIO.U
ICancelOption.ThrowException)
```

Cet exemple supprime le fichier `Test.txt` et l'envoie à la **Corbeille**.

VB

```
My.Computer.FileSystem.DeleteFile _
("C:\test.txt", FileIO.UIOption.OnlyErrorDialogs, FileIO.RecycleOption.SendToRecycleBin, Fi
leIO.UICancelOption.ThrowException)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**

Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : SafeSubWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : supprimer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem, objet
UIOption, énumération
RecycleOption, énumération
UICancelOption, énumération
My.Computer.FileSystem, objet
Microsoft.VisualBasic.FileIO.FileSystem.DeleteFile

My.Computer.FileSystem.DirectoryExists, méthode

Retourne **Boolean** qui indique si le répertoire spécifié existe.

```
' Usage
Dim value As Boolean = My.Computer.FileSystem.DirectoryExists(directory)
' Declaration
Public Function DirectoryExists( _
   ByVal directory As String _
) As Boolean
```

Paramètres

directory

String. Chemin d'accès du répertoire. Obligatoire.

Valeur de retour

True si le répertoire existe ; sinon **False**.

Exceptions

Cette méthode ne lève pas d'exceptions.

Notes

L'appel à **DirectoryExists** requiert FileIOPermission.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.DirectoryExists**.

Pour	Consultez
Déterminer si un répertoire existe	Comment : déterminer si un répertoire existe dans Visual Basic

Exemple

Cet exemple détermine si le répertoire `C:\backup\logs` existe et vérifie ses propriétés.

VB

```
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    Dim logInfo As System.IO.DirectoryInfo
    logInfo = My.Computer.FileSystem.GetDirectoryInfo _
    ("C:\backup\logs")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**

Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer si un fichier existe dans Visual Basic

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet

DirectoryExists

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.Drives, propriété

Retourne une collection en lecture seule de tous les noms de lecteurs disponibles.

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of System.IO.DriveInfo) = My
.Computer.FileSystem.Drives
' Declaration
Public ReadOnly Property Drives As System.Collections.ObjectModel.ReadOnlyCollection(Of Sys
tem.IO.DriveInfo)
```

Valeur de retour

System.Collections.ObjectModel.ReadOnlyCollection (de **System.IO.DriveInfo**)

Exceptions

Cette propriété ne lève pas d'exceptions.

Notes

Cette propriété retourne tous les lecteurs logiques.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.Computer.FileSystem.Drives**.

Pour	**Reportez-vous à**
Déterminer le nom de volume d'un lecteur	Comment : déterminer le nom de volume d'un lecteur dans Visual Basic
Déterminer le format d'un lecteur	Comment : déterminer le format d'un lecteur dans Visual Basic
Déterminer le type d'un lecteur	Comment : déterminer le type d'un lecteur dans Visual Basic
Déterminer l'espace total d'un lecteur	Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Déterminer l'espace disponible d'un lecteur	Comment : déterminer l'espace physique disponible sur un lecteur dans Visual Basic
Déterminer le répertoire racine d'un lecteur	Comment : déterminer le répertoire racine d'un lecteur dans Visual Basic

Exemple

Cet exemple affiche les noms de lecteurs disponibles dans un message.

VB

```
Dim getInfo As System.IO.DriveInfo()
getInfo = System.IO.DriveInfo.GetDrives
For Each info As System.IO.DriveInfo In getInfo
    MsgBox(info.name)
Next
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet
System.Collections.ObjectModel.ReadOnlyCollection
System.IO.DriveInfo
Microsoft.VisualBasic.FileIO.FileSystem.Drives

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.FileExists, méthode

Retourne un **Boolean** indiquant si le fichier spécifié existe.

```
' Usage
Dim value As Boolean = My.Computer.FileSystem.FileExists(file)
' Declaration
Public Function FileExists( _
   ByVal file As String _
) As Boolean
```

Paramètres

file

String. Nom et chemin d'accès du fichier. Obligatoire.

Valeur de retour

Retourne **True** si le fichier existe ; sinon, cette méthode retourne **False**.

Exceptions

La condition ci-dessous peut générer une exception :

- le nom du fichier se termine par une barre oblique inverse (\) (ArgumentException).

Notes

Si l'application ne dispose pas des autorisations nécessaires pour lire le fichier spécifié, la méthode **FileExists** retourne **False**, qu'il existe ou non un chemin d'accès, mais ne lève pas d'exception.

Tâches

Le tableau suivant répertorie un exemple de tâche impliquant la méthode **My.Computer.FileSystem.FileExists**.

Pour	Reportez-vous à
Déterminer si un fichier existe.	Comment : déterminer si un fichier existe dans Visual Basic

Exemple

Cet exemple vérifie si oui ou non le fichier `Check.txt` existe et fournit les informations dans un message.

VB

```
If My.Computer.FileSystem.FileExists("c://Check.txt") Then
    MsgBox("File found.")
Else
    MsgBox("File not found.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**

Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer si un répertoire existe dans Visual Basic

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet

FileExists

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.FindInFiles, méthode

Retourne une collection en lecture seule de chaînes qui représentent le nom des fichiers contenant le texte spécifié.

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.FindInFiles(directory ,containsText ,ignoreCase ,searchType)
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.FindInFiles(directory ,containsText ,ignoreCase ,searchType ,fileWildcards)
' Declaration
Public Function FindInFiles( _
   ByVal directory As String, _
   ByVal containsText As String, _
   ByVal ignoreCase As Boolean, _
   ByVal searchType As SearchOption _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
' -or-
Public Function FindInFiles( _
   ByVal directory As String, _
   ByVal containsText As String, _
   ByVal ignoreCase As Boolean, _
   ByVal searchType As SearchOption, _
   ByVal fileWildcards As String() _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
```

Paramètres

directory

String. Répertoire à rechercher. Obligatoire.

containsText

String. Texte de recherche. Obligatoire.

ignoreCase

Boolean. Si la recherche doit respecter ou non la casse. La valeur par défaut est **True**. Obligatoire.

searchType

SearchOption, énumération. Inclure, ou non, les sous-dossiers. La valeur par défaut est **SearchOption.SearchTopLevelOnly**. Obligatoire.

fileWildcards

Tableau de type **String**. Modèle à faire correspondre. Obligatoire.

Valeur de retour

Collection en lecture seule de type **String**.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par `\\.\`) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe vers un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct

(NotSupportedException).

- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Notes

Une collection vide est retournée si aucun fichier correspondant au modèle spécifié n'est trouvé.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.FindInFiles**.

Pour	Consultez
Rechercher dans un répertoire des fichiers contenant une chaîne spécifique	Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Exemple

Cet exemple recherche dans le répertoire `C:\TestDir` des fichiers contenant la chaîne `"sample string"` et affiche les résultats dans `ListBox1`.

VB

```
Dim list As System.Collections.ObjectModel.ReadOnlyCollection _
(Of String)
list = My.Computer.FileSystem.FindInFiles("C:\TestDir", _
 "sample string", True, FileIO.SearchOption.SearchTopLevelOnly)
For Each name As String In list
    ListBox1.Items.Add(name)
Next
```

Pour que cet exemple fonctionne, le projet doit contenir un **ListBox** nommé `ListBox1`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description

FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet

SearchOption, énumération

System.Collections.ObjectModel.ReadOnlyCollection

Microsoft.VisualBasic.FileIO.FileSystem.FindInFiles

My.Computer.FileSystem.GetDirectories, méthode

Retourne une collection de chaînes représentant les chemins d'accès de sous-répertoires présents dans un répertoire défini.

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetDirectories(directory)
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetDirectories(directory ,searchType ,wildcards)
' Declaration
Public Function GetDirectories( _
   ByVal directory As String _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
' -or-
Public Function GetDirectories( _
   ByVal directory As String, _
   ByVal searchType As SearchOption, _
   ByVal wildcards As String() _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
```

Paramètres

directory

String. Nom et chemin d'accès du répertoire. Obligatoire.

searchType

SearchOption, énumération. Inclure, ou non, les sous-dossiers. La valeur par défaut est **SearchOption.SearchTopLevelOnly**. Obligatoire.

wildcards

String. Modèle pour la correspondance de noms. Obligatoire.

Valeur de retour

Collection en lecture seule de **String**.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Un ou plusieurs des caractères génériques spécifiés a la valeur **Nothing**, est une chaîne vide ou contient uniquement des espaces (**ArgumentNullException**).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe vers un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Notes

Vous pouvez utiliser le paramètre *wildcards* pour spécifier un modèle spécifique. Si vous souhaitez inclure le contenu des

sous-répertoires dans la recherche, affectez **SearchAllSubDirectories** au paramètre *searchType*.

Une collection vide est retournée si aucun répertoire correspondant au modèle spécifié n'est trouvé.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.GetDirectories**.

Pour	Consultez
Répertorier les sous-répertoires avec un modèle spécifique	Comment : rechercher des sous-répertoires avec un modèle spécifique dans Visual Basic

Exemple

L'exemple suivant retourne tous les répertoires dans la structure de répertoire qui contiennent le mot `Logs` dans leurs noms et les ajoute dans `ListBox1`.

VB

```
For Each foundDirectory As String In _
My.Computer.FileSystem.GetDirectories _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
True, "*Logs*")
    ListBox1.Items.Add(foundDirectory)
Next
```

Cet exemple implique que vous disposiez dans votre formulaire d'un **ListBox** nommé `ListBox1`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

SearchOption, énumération
System.Collections.ObjectModel.ReadOnlyCollection
Microsoft.VisualBasic.FileIO.FileSystem.GetDirectories

My.Computer.FileSystem.GetDirectoryInfo, méthode

Retourne un objet DirectoryInfo pour le chemin d'accès spécifié.

```
' Usage
Dim value As System.IO.DirectoryInfo = My.Computer.FileSystem.GetDirectoryInfo(directory)
' Declaration
Public Function GetDirectoryInfo( _
   ByVal directory As String _
) As System.IO.DirectoryInfo
```

Paramètres

directory

String. Chemin d'accès du répertoire. Obligatoire.

Valeur de retour

DirectoryInfo

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès du répertoire n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès du répertoire n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès du répertoire dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès du répertoire contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès du répertoire (SecurityException).

Notes

Si le répertoire n'existe pas, une exception n'est pas levée tant qu'une propriété de l'objet **DirectoryInfo** n'est pas accédée pour la première fois.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.GetDirectoryInfo**.

Pour	Consultez
Déterminer la date de création d'un répertoire	Comment : déterminer l'heure de création d'un répertoire dans Visual Basic
Déterminer si un répertoire est en lecture seule	Comment : déterminer si un répertoire est en lecture seule dans Visual Basic

Exemple

Cet exemple obtient un objet **DirectoryInfo** pour le répertoire `C:\Documents and Settings` et affiche l'heure de création, l'heure de dernier accès et l'heure de dernière écriture du répertoire.

VB

```
Dim getInfo As System.IO.DirectoryInfo
getInfo = My.Computer.FileSystem.GetDirectoryInfo _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
MsgBox("The directory was created at " & getInfo.CreationTime)
MsgBox("The directory was last accessed at " & getInfo.LastAccessTime)
MsgBox("The directory was last written to at " & getInfo.LastWriteTime)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet
System.IO.DirectoryInfo
GetDirectoryInfo

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.GetDriveInfo, méthode

Retourne un objet DriveInfo pour le lecteur spécifié.

```
' Usage
Dim value As System.IO.DriveInfo = My.Computer.FileSystem.GetDriveInfo(drive)
' Declaration
Public Function GetDriveInfo( _
   ByVal drive As String _
) As System.IO.DriveInfo
```

Paramètres

drive

String. Lecteur à examiner. Obligatoire.

Valeur de retour

DriveInfo

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de dossier du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Notes

La classe **DriveInfo** modèle un lecteur et fournit des méthodes et des propriétés pour demander des informations sur le lecteur. Utilisez **DriveInfo** pour déterminer les lecteurs disponibles et leur type. Vous pouvez également interroger la propriété pour déterminer la capacité et l'espace libre disponible sur le lecteur.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.GetDriveInfo**.

Pour	**Consultez**
Déterminer le nom de volume d'un lecteur	Comment : déterminer le nom de volume d'un lecteur dans Visual Basic
Déterminer le type d'un lecteur	Comment : déterminer le type d'un lecteur dans Visual Basic
Déterminer l'espace total d'un lecteur	Comment : déterminer l'espace total d'un lecteur dans Visual Basic
Déterminer l'espace physique libre d'un lecteur	Comment : déterminer l'espace physique disponible sur un lecteur dans Visual Basic
Déterminer le dossier racine d'un lecteur	Comment : déterminer le répertoire racine d'un lecteur dans Visual Basic

Exemple

Cet exemple obtient un objet **DriveInfo** pour le lecteur C et l'utilise pour afficher des informations sur le lecteur.

VB

```
Dim getInfo As System.IO.DriveInfo
getInfo = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox("The drive's type is " & getInfo.DriveType)
MsgBox("The drive has " & getInfo.TotalFreeSpace & " bytes free.")
```

Pour plus d'informations sur les différents types de lecteur, consultez DriveType.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet
System.IO.DriveInfo
GetDriveInfo
DriveType

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.GetFileInfo, méthode

Retourne un objet FileInfo pour le fichier spécifié.

```
' Usage
Dim value As System.IO.FileInfo = My.Computer.FileSystem.GetFileInfo(file)
' Declaration
Public Function GetFileInfo( _
   ByVal file As String _
) As System.IO.FileInfo
```

Paramètres

file

String. Nom et chemin d'accès du fichier. Obligatoire.

Valeur de retour

FileInfo

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le nom du chemin d'accès est incorrect. Il contient par exemple des caractères non valides ou se compose uniquement d'un espace blanc (ArgumentException).
- Le nom du fichier contient une marque de barre oblique finale (**ArgumentException**).
- Le fichier n'existe pas ou a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès comporte un signe deux-points au milieu de la chaîne (NotSupportedException).
- Le chemin d'accès est trop long (PathTooLongException
- L'utilisateur n'a pas les autorisations nécessaires (SecurityException).
- L'utilisateur ne dispose pas d'un accès à la liste de contrôle d'accès (ACL, Access Control List) au fichier (UnauthorizedAccessException).

Notes

Une exception n'est pas levée si le fichier n'existe pas ; en revanche, elle sera levée lorsque vous accédez pour la première fois aux propriétés de l'objet.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.GetFileInfo**.

Pour	Consultez
Obtenir des informations sur un fichier	Comment : obtenir des informations à propos d'un fichier dans Visual Basic
Déterminer le nom et le chemin d'accès d'un fichier	Comment : analyser des chemins d'accès dans Visual Basic

Exemple

Cet exemple récupère un objet **System.IO.FileInfo** pour le fichier `MyLogFile.log` et l'utilise pour déterminer le nom complet, l'heure du dernier accès et la longueur du fichier.

VB

```
Dim information As System.IO.FileInfo
information = My.Computer.FileSystem.GetFileInfo("C:\MyLogFile.log")
MsgBox("The file's full name is " & information.FullName & ".")
MsgBox("Last access time is " & information.LastAccessTime & ".")
```

```
MsgBox("The length is " & information.Length & ".")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet

System.IO.FileInfo

Microsoft.VisualBasic.FileIO.FileSystem.GetFileInfo(System.String)

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.GetFiles, méthode

Retourne une collection en lecture seule de chaînes qui représentent le nom des fichiers d'un répertoire.

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetFiles(directory)
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetFiles(directory ,searchType ,wildcards)
' Declaration
Public Function GetFiles( _
   ByVal directory As String _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
' -or-
Public Function GetFiles( _
   ByVal directory As String, _
   ByVal searchType As SearchOption, _
   ByVal wildcards As String() _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
```

Paramètres

directory

String. Répertoire à rechercher. Obligatoire.

searchType

SearchOption, énumération. Inclure, ou non, les sous-dossiers. La valeur par défaut est **SearchOption.SearchTopLevelOnly**. Obligatoire.

wildcards

String. Modèle à mettre en correspondance. Obligatoire.

Valeur de retour

Collection en lecture seule de chaînes.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- *directory* n'existe pas (DirectoryNotFoundException).
- *directory* pointe vers un fichier existant (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (UnauthorizedAccessException).

Notes

Une collection vide est retournée si aucun fichier correspondant au modèle spécifié n'est trouvé.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.GetFiles**.

Pour	Consultez
Obtenir la collection de fichiers d'un répertoire	Comment : placer la collection de fichiers dans un répertoire dans Visual Basic
Rechercher des fichiers d'un modèle spécifique dans un répertoire	Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic

Exemple

L'exemple suivant retourne tous les fichiers du répertoire et les ajoute dans `ListBox1`.

VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
    ListBox1.Items.Add(foundFile)
Next
```

Cet exemple implique que vous disposiez dans votre formulaire d'un **ListBox** nommé `ListBox1`.

Cet exemple retourne tous les fichiers du répertoire dont l'extension est `.dll` et les ajoute dans `ListBox1`.

VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments, True, "*.dll")
    ListBox1.Items.Add(foundFile)
Next
```

Cet exemple implique que vous disposiez dans votre formulaire d'un **ListBox** nommé `ListBox1`.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : rechercher des fichiers avec un modèle spécifique dans Visual Basic

Comment : placer la collection de fichiers dans un répertoire dans Visual Basic

Référence

My.Computer.FileSystem, objet

SearchOption, énumération

System.Collections.ObjectModel.ReadOnlyCollection

Microsoft.VisualBasic.FileIO.FileSystem.GetFiles

My.Computer.FileSystem.GetName, méthode

Analyse le nom de fichier en dehors du chemin d'accès fourni.

```
' Usage
Dim value As String = My.Computer.FileSystem.GetFileName(path)
' Declaration
Public Function GetFileName( _
   ByVal path As String _
) As String
```

Paramètres

path

Obligatoire. Chemin d'accès à analyser. **String**.

Valeur de retour

String.

Exceptions

Cette méthode ne lève pas d'exceptions.

Notes

Ceci est une opération de chaînes ; **FileSystem** n'est pas examiné.

La méthode **GetName** ignore une marque de barre oblique présente à la fin du chemin d'accès.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.GetFileName**.

Pour	Consultez
Analyser un chemin d'accès	Comment : analyser des chemins d'accès dans Visual Basic

Exemple

L'exemple suivant analyse un chemin d'accès et retourne le nom du fichier.

VB

```
MsgBox("The filename is: " & _
My.Computer.FileSystem.GetName("C:\testdirectory\testfile"))
```

Remplacez le chemin d'accès `C:\Testdirectory\Testfile` par celui à analyser.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**

Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.FileSystem, objet

GetFileName

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.GetParentPath, méthode

Retourne le chemin d'accès parent du chemin fourni.

```
' Usage
Dim value As String = My.Computer.FileSystem.GetParentPath(path)
' Declaration
Public Function GetParentPath( _
   ByVal path As String _
) As String
```

Paramètres

path

String. Chemin d'accès à examiner. Requis.

Valeur de retour

String.

Exceptions

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'a pas de chemin parent parce qu'il s'agit d'un chemin d'accès racine (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Notes

Il s'agit d'une opération de chaîne ; le système de fichiers n'est pas examiné.

Exemple

Cet exemple obtient le chemin d'accès parent pour `C:\Backups\Tmp\Test`.

VB

```
Dim strPath As String
strPath = My.Computer.FileSystem.GetParentPath("C:\backups\tmp\test")
MsgBox(strPath)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**

Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem, objet

GetParentPath

Autres ressources

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

My.Computer.FileSystem.GetTempFileName, méthode

Crée un fichier temporaire de zéro octet nommé de manière unique sur le disque et retourne le chemin d'accès complet de ce fichier.

```
' Usage
Dim value As String = My.Computer.FileSystem.GetTempFileName()
' Declaration
Public Function GetTempFileName() As String
```

Valeur de retour

String contenant le chemin d'accès complet du fichier temporaire.

Notes

Cette méthode peut être utilisée pour créer un fichier temporaire.

Exemple

Cet exemple crée un fichier temporaire et retourne son chemin d'accès.

VB

```
MsgBox("The file is located at " & _
My.Computer.FileSystem.GetTempFileName)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

Microsoft.VisualBasic.FileIO.FileSystem.GetTempFileName

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.MoveDirectory, méthode

Déplace un répertoire d'un emplacement à l'autre.

```
' Usage
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName)
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName ,overwri
te)
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI)
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI
,onUserCancel)

' Declaration
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String _
)
' -or-
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

sourceDirectoryName

String. Chemin d'accès au répertoire à déplacer. Obligatoire.

destinationDirectoryName

String. Chemin d'accès au répertoire dans lequel le répertoire source est déplacé. Obligatoire.

overwrite

Boolean. Spécifie si les répertoires existants doivent être remplacés. La valeur par défaut est **False**. Obligatoire.

showUI

UIOption. Spécifie s'il faut effectuer un suivi visuel de l'avancement de l'opération. La valeur par défaut est **UIOption.OnlyErrorDialogs**. Obligatoire.

onUserCancel

Énumération UICancelOption. Spécifie si oui ou non une exception est levée lorsque l'utilisateur annule l'opération. La valeur par défaut est **UICancelOption.ThrowException**. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- le chemin d'accès est une chaîne de longueur nulle, ne contient que des espaces blancs, contient des caractères non valides ou est un chemin d'accès de périphérique (qui commence par \\. \) (ArgumentException) ;

- le chemin d'accès est **Nothing** (ArgumentNullException) ;
- la source n'est pas valide (DirectoryNotFoundException) ;
- la source est un répertoire racine (IOException) ;
- le chemin d'accès combiné pointe sur un fichier existant (**IOException**) ;
- le chemin source et le chemin cible sont identiques (**IOException**) ;
- le fichier existe déjà et *verwrite* a la valeur **False** (**IOException**) ;
- *onUserCancel* a la valeur **ThrowException** et un sous-répertoire du fichier ne peut pas être copié (**IOException**) ;
- l'opération est cyclique (InvalidOperationException) ;
- un nom de fichier ou de répertoire du chemin d'accès contient le signe deux-points (:) (NotSupportedException) ;
- **UICancelOption** a la valeur **ThrowException** et l'utilisateur annule l'opération, ou encore l'opération ne peut pas être effectuée (OperationCanceledException) ;
- le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException) ;
- **UICancelOption** a la valeur **ThrowException** et l'utilisateur n'a pas les autorisations nécessaires (SecurityException) ;
- l'utilisateur n'est pas autorisé à créer un fichier (UnauthorizedAccessException).

Notes

Si vous tentez de déplacer un répertoire vers un répertoire qui n'existe pas, la structure cible sera créée.

Tâches

Le tableau suivant répertorie un exemple de tâche impliquant la méthode **My.Computer.FileSystem.MoveDirectory**.

Pour	**Reportez-vous à**
Déplacer un répertoire	Comment : déplacer un répertoire dans Visual Basic

Exemple

Cet exemple déplace `Directory1` dans `Directory2`.

VB

```
My.Computer.FileSystem.MoveDirectory("C:\Directory1", "C:\Directory2")
```

Cet exemple déplace `Directory1` à l'intérieur de `Directory2`, remplaçant le répertoire s'il existe déjà.

VB

```
My.Computer.FileSystem.MoveDirectory("C:\Directory1", "C:\Directory2", _
True)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**

Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au Presse-papiers. Énumération associée : SafeSubWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet
UIOption, énumération
UICancelOption, énumération
My.Computer.FileSystem.MoveFile, méthode
My.Computer.FileSystem.CopyDirectory, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.MoveFile, méthode

Déplace un fichier vers un nouvel emplacement.

```
' Usage
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName)
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName ,overwrite)
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName ,showUI)
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName ,showUI ,onUserCancel)
' Declaration
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

sourceFileName

String. Chemin d'accès au fichier à déplacer. Obligatoire.

destinationFileName

String. Chemin d'accès au répertoire dans lequel le fichier doit être déplacé. Obligatoire.

overwrite

Boolean. Spécifie s'il faut remplacer des fichiers existants. La valeur par défaut est **False**. Obligatoire.

showUI

Énumération UIOption. Spécifie s'il faut effectuer un suivi visuel de l'avancement de l'opération. La valeur par défaut est **UIOption.OnlyErrorDialogs**. Obligatoire.

onUserCancel

Énumération UICancelOption. Spécifie si oui ou non une exception est levée lorsque l'utilisateur annule l'opération. La valeur par défaut est **UICancelOption.ThrowException**. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- le nom de fichier se termine par une barre oblique inverse (**ArgumentException**) ;

- le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException) ;
- *destinationFileName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**) ;
- le fichier source n'est pas valide ou n'existe pas (FileNotFoundException) ;
- le chemin d'accès combiné pointe sur un répertoire existant, le fichier de destination existe et *overwrite* a la valeur **False**, un fichier du répertoire cible portant le même nom est en cours d'utilisation, ou l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (IOException) ;
- un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException) ;
- *onUserCancel* a la valeur **ThrowException** et l'utilisateur a annulé l'opération ou une erreur E/S non spécifiée se produit (OperationCanceledException) ;
- le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException) ;
- l'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException) ;
- l'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException) ;

Notes

Si la structure cible n'existe pas, elle est créée.

La méthode **MoveFile** conserve les entrées du contrôle d'accès (ACE, Access Control Entries) uniquement lors du déplacement du fichier dans le même volume. Cela inclut des entrées du contrôle d'accès (ACE) héritées, qui deviennent des entrées directes lorsqu'elles sont déplacées (les entrées du contrôle d'accès directes ont la priorité sur celles qui sont héritées). Si un fichier est déplacé entre les volumes, les entrées du contrôle d'accès ne seront pas copiées.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.MoveFile**.

Pour	**Reportez-vous à**
Déplacer un fichier	Comment : déplacer un fichier dans Visual Basic
Déplacer une collection de fichiers	Comment : déplacer une collection de fichiers dans Visual Basic
Déplacer le contenu d'un répertoire	Comment : déplacer le contenu d'un répertoire dans Visual Basic

Exemple

Cet exemple déplace le fichier `Test.txt` de `TestDir1` vers `TestDir2`.

VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", "C:\TestDir2\test.txt")
```

Cet exemple déplace le fichier `Test.txt` de `TestDir1` vers `TestDir2` et le renomme `Test2.txt`.

VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", "C:\TestDir2\test2.txt")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	**Description**
EnvironmentPermission	Contrôle la possibilité d'accéder à toutes les variables d'environnement. Énumération associée : Unrestricted.
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : **Unrestricted**.
RegistryPermission	Contrôle la possibilité d'accéder aux variables du Registre. Énumération associée : **Unrestricted**.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au Presse-papiers. Énumération associée : SafeSubWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

UIOption, énumération

UICancelOption, énumération

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.OpenTextFieldParser, méthode

La méthode **OpenTextFieldParser** vous permet de créer un objet TextFieldParser et d'analyser efficacement les fichiers texte structurés, comme les journaux. L'objet **TextFieldParser** peut être utilisé pour lire des fichiers délimités et à largeur fixe.

```
' Usage
Dim value As TextFieldParser = My.Computer.FileSystem.OpenTextFieldParser(file)
Dim value As TextFieldParser = My.Computer.FileSystem.OpenTextFieldParser(file ,delimiters)
Dim value As TextFieldParser = My.Computer.FileSystem.OpenTextFieldParser(file ,fieldWidths
)
' Declaration
Public Function OpenTextFieldParser( _
   ByVal file As String _
) As TextFieldParser
' -or-
Public Function OpenTextFieldParser( _
   ByVal file As String, _
   ByVal delimiters As String() _
) As TextFieldParser
' -or-
Public Function OpenTextFieldParser( _
   ByVal file As String, _
   ByVal fieldWidths As Integer() _
) As TextFieldParser
```

Paramètres

file

String. Fichier à ouvrir avec **TextFieldParser**. Obligatoire.

delimiters

String(). Séparateurs pour les champs. Obligatoire.

fieldWidths

Integer(). Largeurs des champs. Obligatoire.

Valeur de retour

TextFieldParser

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- le nom de fichier se termine par une barre oblique finale (**ArgumentException**) ;
- le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException) ;
- le fichier spécifié n'existe pas (FileNotFoundException) ;
- le fichier spécifié est en cours d'utilisation (IOException) ;
- le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException) ;
- un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException) ;
- l'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException) ;
- une ligne ne peut pas être analysée avec le format spécifié (MalformedLineException). Le message d'exception spécifie la ligne qui provoque l'exception, tandis que le texte contenu dans la ligne est assigné à la

propriété TextFieldParser.ErrorLine.

- l'utilisateur n'a pas les autorisations suffisantes pour accéder au fichier (UnauthorizedAccessException).

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.OpenTextFieldParser**.

Pour	**Reportez-vous à**
Lire un fichier texte délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier texte à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Lire un fichier texte avec plusieurs formats	Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Exemple

Cet exemple ouvre `reader` **TextFieldParser** et l'utilise pour lire `C:\TestFolder1\Test1.txt`.

VB

```
Dim reader As Microsoft.VisualBasic.FileIO.TextFieldParser
reader = My.Computer.FileSystem.OpenTextFieldParser _
("C:\TestFolder1\test1.txt")
reader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
reader.delimiters = New String() {","}
Dim currentRow As String()
While Not reader.EndOfData
    Try
        currentRow = reader.ReadFields()
        Dim currentField As String
        For Each currentField In currentRow
            MsgBox(currentField)
        Next
    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
        MsgBox("Line " & ex.Message & _
        "is not valid and will be skipped.")
    End Try
End While
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**

Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
SecurityPermission	Décrit un ensemble d'autorisations de sécurité appliqué au code. Énumération associée : ControlEvidence.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : lire des fichiers texte dans Visual Basic

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem, objet

TextFieldParser, objet

OpenTextFieldParser

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Autres ressources

Lecture à partir de fichiers en Visual Basic

My.Computer.FileSystem.OpenTextFileReader, méthode

Ouvre StreamReader.

```
' Usage
Dim value As System.IO.StreamReader = My.Computer.FileSystem.OpenTextFileReader(file)
Dim value As System.IO.StreamReader = My.Computer.FileSystem.OpenTextFileReader(file ,encod
ing)
' Declaration
Public Function OpenTextFileReader( _
   ByVal file As String _
) As System.IO.StreamReader
' -or-
Public Function OpenTextFileReader( _
   ByVal file As String, _
   ByVal encoding As System.Text.Encoding _
) As System.IO.StreamReader
```

Paramètres

file

String. Fichier à lire. Obligatoire.

encoding

Encoding. La valeur par défaut est UTF8. Requis.

Valeur de retour

StreamReader

Exceptions

Les conditions ci-dessous peuvent générer une exception.

- Le nom de fichier se termine par une barre oblique inverse (\) (ArgumentException).
- Le fichier spécifié est introuvable (FileNotFoundException).
- L'utilisateur n'a pas les autorisations nécessaires pour lire le fichier (SecurityException).

Notes

Seuls les fichiers texte peuvent être lus avec **StreamReader**.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.OpenTextFileReader**.

Pour	Consultez
Ouvrir un fichier avec **StreamReader**	Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

Exemple

Cet exemple ouvre le fichier `Testfile.txt`, en lit une ligne et l'affiche dans **MessageBox**.

VB

```
Dim fileReader As System.IO.StreamReader
fileReader = _
My.Computer.FileSystem.OpenTextFileReader("C:\\testfile.txt")
Dim stringReader As String
stringReader = fileReader.ReadLine()
MsgBox("The first line of the file is " & stringReader)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

Référence

My.Computer.FileSystem, objet

System.Text.Encoding

System.IO.StreamReader

Microsoft.VisualBasic.FileIO.FileSystem.OpenTextFileReader

Autres ressources

Lecture à partir de fichiers en Visual Basic

My.Computer.FileSystem.OpenTextFileWriter, méthode

Ouvre StreamWriter.

```
' Usage
Dim value As System.IO.StreamWriter = My.Computer.FileSystem.OpenTextFileWriter(file ,appen
d)
Dim value As System.IO.StreamWriter = My.Computer.FileSystem.OpenTextFileWriter(file ,appen
d ,encoding)
' Declaration
Public Function OpenTextFileWriter( _
   ByVal file As String, _
   ByVal append As Boolean _
) As System.IO.StreamWriter
' -or-
Public Function OpenTextFileWriter( _
   ByVal file As String, _
   ByVal append As Boolean, _
   ByVal encoding As System.Text.Encoding _
) As System.IO.StreamWriter
```

Paramètres

file

String. Fichier dans lequel écrire. Obligatoire.

append

Boolean. Spécifie s'il faut ajouter ou remplacer les informations contenues dans le fichier. La valeur par défaut est **False**. Obligatoire.

encoding

Encoding. Codage à utiliser pour écrire dans le fichier. La valeur par défaut est UTF8. Requis.

Valeur de retour

StreamWriter

Exceptions

La condition ci-dessous peut lever une exception :

- Le nom du fichier se termine par une barre oblique finale (ArgumentException).

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.FileSystem.OpenTextFileWriter**.

Pour	Consultez
Écrire du texte dans un fichier avec **StreamWriter**	Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Exemple

Cet exemple ouvre **StreamWriter** à l'aide de la méthode **My.Computer.FileSystem.OpenTextFileWriter** et l'utilise pour écrire une chaîne dans un fichier texte à l'aide de la méthode **WriteLine** de la classe **StreamWriter**.

VB

```
Dim file As System.IO.StreamWriter
file = My.Computer.FileSystem.OpenTextFileWriter("c:\test.txt", True)
file.WriteLine("Here is the first string.")
file.Close()
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : écrire du texte dans des fichiers à l'aide de Streamwriter dans Visual Basic

Référence

My.Computer.FileSystem, objet
System.Text.Encoding
System.IO.StreamWriter
Microsoft.VisualBasic.FileIO.FileSystem.OpenTextFileWriter

My.Computer.FileSystem.ReadAllBytes, méthode

Retourne le contenu d'un fichier sous forme de tableau d'octets.

```
' Usage
Dim value As Byte() = My.Computer.FileSystem.ReadAllBytes(file)
' Declaration
Public Function ReadAllBytes( _
   ByVal file As String _
) As Byte()
```

Paramètres

file

String. Fichier à lire. Obligatoire.

Valeur de retour

Tableau **Byte** qui contient le contenu du fichier.

Exceptions

Les conditions ci-dessous peuvent lever une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le nom du fichier se termine par une barre oblique finale (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier n'existe pas (FileNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur d'E/S s'est produite (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- La mémoire est insuffisante pour écrire la chaîne dans la mémoire tampon (OutOfMemoryException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Notes

La méthode **ReadAllBytes** de l'objet **My.Computer.FileSystem** vous permet de lire un fichier binaire. Le contenu du fichier est retourné sous forme de tableau d'octets.

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut qu'un fichier nommé Form1.vb ne soit pas un fichier source Visual Basic. Vérifiez toutes les entrées avant d'utiliser les données dans votre application.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.ReadAllBytes**.

Pour	Consultez
Lire un fichier binaire	Comment : lire des fichiers binaires dans Visual Basic

Exemple

Cet exemple lit le fichier `C:/Documents and Settings/selfportrait.jpg`.

VB

```
My.Computer.FileSystem.ReadAllBytes _
("C:/Documents and Settings/selfportrait.jpg")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

ReadAllBytes

Autres ressources

Lecture à partir de fichiers en Visual Basic

My.Computer.FileSystem.ReadAllText, méthode

Retourne le contenu d'un fichier texte sous forme de **String**.

```
' Usage
Dim value As String = My.Computer.FileSystem.ReadAllText(file)
Dim value As String = My.Computer.FileSystem.ReadAllText(file ,encoding)
' Declaration
Public Function ReadAllText( _
   ByVal file As String _
) As String
' -or-
Public Function ReadAllText( _
   ByVal file As String, _
   ByVal encoding As System.Text.Encoding _
) As String
```

Paramètres

file

String. Nom et chemin d'accès du fichier à lire. Obligatoire.

encoding

System.Text.Encoding. Codage de caractères à utiliser pour lire le fichier. Obligatoire. La valeur par défaut est UTF-8.

Valeur de retour

String qui contient le contenu du fichier.

Exceptions

Le fichier n'a peut-être pas le contenu attendu, et les méthodes utilisées pour lire le fichier peuvent échouer.

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le nom du fichier se termine par une barre oblique finale (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le fichier n'existe pas (FileNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur d'E/S s'est produite (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- La mémoire est insuffisante pour écrire la chaîne dans la mémoire tampon (OutOfMemoryException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Notes

La méthode **ReadAllText** de l'objet **My.Computer.FileSystem** vous permet de lire un fichier texte. Le contenu du fichier est retourné sous forme de chaîne.

Le codage de fichier peut être spécifié si le contenu du fichier utilise un codage ASCII ou UTF-8. Si vous lisez un fichier contenant des caractères étendus, vous devez spécifier le codage de fichier.

Ne vous basez pas sur le nom d'un fichier pour en déterminer le contenu. Par exemple, il se peut qu'un fichier nommé Form1.vb ne soit pas un fichier source Visual Basic. Vérifiez toutes les entrées avant d'utiliser les données dans votre

application.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.ReadAllText**.

Pour	Consultez
Lire un fichier texte.	Comment : lire des fichiers texte dans Visual Basic

Exemple

Cet exemple lit le contenu de `Test.txt` dans une chaîne puis l'affiche dans un message.

VB

```
Dim reader As String
reader = My.Computer.FileSystem.ReadAllText("C:\test.txt")
MsgBox(reader)
```

Cet exemple lit le contenu du fichier ASCII `Test.txt` dans une chaîne puis l'affiche dans un message.

VB

```
Dim reader As String
reader = My.Computer.FileSystem.ReadAllText("C:\test.txt", _
   System.Text.Encoding.ASCII)
MsgBox(reader)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)
Dépannage : lecture et écriture dans des fichiers texte
Procédure pas à pas : manipulation de fichiers et de répertoires en Visual Basic

Référence

My.Computer.FileSystem, objet
System.Text.Encoding
Microsoft.VisualBasic.FileIO.FileSystem.ReadAllText
StreamReader

Concepts

Codages de fichiers

Autres ressources

Lecture à partir de fichiers en Visual Basic

My.Computer.FileSystem.RenameDirectory, méthode

Renomme un répertoire.

```
' Usage
My.Computer.FileSystem.RenameDirectory(directory ,newName)
' Declaration
Public Sub RenameDirectory( _
   ByVal directory As String, _
   ByVal newName As String _
)
```

Paramètres

directory

String. Chemin d'accès et nom du répertoire à renommer. Obligatoire.

newName

String. Nouveau nom du répertoire. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le paramètre *newName* contient des informations relatives au chemin d'accès (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le paramètre *newName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le répertoire source n'est pas valide ou n'existe pas (DirectoryNotFoundException).
- Il existe un fichier ou un répertoire portant le nom spécifié dans *newName* (IOException).
- Le répertoire est un répertoire racine (**IOException**).
- Le chemin d'accès dépasse 248 caractères (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).

Notes

Cette méthode ne peut pas être utilisée pour déplacer un répertoire ; utilisez la méthode **MoveDirectory** pour déplacer et renommer le répertoire.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.FileSystem.RenameDirectory**.

Pour	Consultez
Renommer un répertoire	Comment : renommer un répertoire dans Visual Basic

Exemple

Cet exemple renomme le répertoire `Test` en `SecondTest`.

VB

```
My.Computer.FileSystem.RenameDirectory("C:MyDocuments\Test", "SecondTest")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.MoveDirectory, méthode

RenameDirectory

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.RenameFile, méthode

Renomme un fichier.

```
' Usage
My.Computer.FileSystem.RenameFile(file ,newName)
' Declaration
Public Sub RenameFile( _
   ByVal file As String, _
   ByVal newName As String _
)
```

Paramètres

file

String. Fichier à renommer. Obligatoire.

newName

String. Nouveau nom de fichier. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le paramètre *newName* contient des informations relatives au chemin d'accès ou se termine par une barre oblique inverse (\) (**ArgumentException**).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException).
- Le paramètre *newName* a la valeur **Nothing** ou est une chaîne vide (**ArgumentNullException**).
- Le fichier source n'est pas valide ou n'existe pas (FileNotFoundException).
- Il existe un fichier ou un répertoire portant le nom spécifié dans *newName* (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).
- L'utilisateur n'a pas l'autorisation requise (UnauthorizedAccessException).

Notes

Cette méthode ne peut pas être utilisée pour déplacer un fichier ; utilisez la méthode My.Computer.FileSystem.MoveFile pour déplacer et renommer le fichier.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.FileSystem.RenameFile**.

Pour	**Consultez**
Renommer un fichier	Comment : renommer un fichier dans Visual Basic

Exemple

Cet exemple renomme le fichier `Test.txt` dans `SecondTest.txt`.

VB

```
My.Computer.FileSystem.RenameFile("C:\Test.txt", "SecondTest.txt")
```

Remplacez "`C:\Test.txt`" par le chemin d'accès et le nom du fichier que vous souhaitez renommer.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

Microsoft.VisualBasic.FileIO.FileSystem.RenameFile(System.String,System.String)

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

My.Computer.FileSystem.SpecialDirectories, propriété

Obtient un objet qui fournit des propriétés d'accès aux répertoires communément référencés.

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.SpecialDirectoriesProxy = My.Computer.FileSys
tem.SpecialDirectories
' Declaration
Public ReadOnly Property SpecialDirectories As MyServices.SpecialDirectoriesProxy
```

Valeur de retour

Cette propriété retourne l'objet My.Computer.FileSystem.SpecialDirectories pour l'ordinateur.

Notes

Cette propriété fournit un accès facile à l'objet My.Computer.FileSystem.SpecialDirectories.

Exemple

Cet exemple retourne et affiche le chemin d'accès du répertoire Desktop de l'utilisateur.

VB

```
Dim filePath As String
filePath = My.Computer.FileSystem.SpecialDirectories.Desktop
MsgBox(filePath)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.FileSystem, objet

My.Computer.FileSystem.SpecialDirectories, objet

My.Computer.FileSystem.WriteAllBytes, méthode

Écrit des données dans un fichier binaire.

```
' Usage
My.Computer.FileSystem.WriteAllBytes(file ,data ,append)
' Declaration
Public Sub WriteAllBytes( _
   ByVal file As String, _
   ByVal data As Byte(), _
   ByVal append As Boolean _
)
```

Paramètres

file

String. Chemin d'accès et nom du fichier dans lequel écrire. Obligatoire.

data

Byte Données à écrire dans le fichier. Obligatoire.

append

Boolean. Si besoin d'ajouter ou de remplacer des données. La valeur par défaut est **False**. Obligatoire.

Exceptions

Les conditions ci-dessous peuvent générer une exception.

- Le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException).
- Le chemin d'accès n'est pas valide, car il a la valeur **Nothing** ou se termine par une barre oblique inverse (\) (ArgumentNullException).
- Le paramètre *file* pointe vers un chemin d'accès qui n'existe pas (FileNotFoundException ou DirectoryNotFoundException).
- Le fichier est utilisé par un autre processus, ou une erreur d'E/S s'est produite (IOException).
- Le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException).
- Un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException).
- L'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Notes

Si le paramètre *append* a la valeur **True**, il ajoute les données au fichier ; sinon, les données du fichier sont remplacées.

Si le chemin d'accès spécifié, à l'exclusion du nom de fichier, n'est pas valide, une exception **DirectoryNotFoundException** est levée. Si le chemin d'accès est valide mais que le fichier n'existe pas, ce dernier est créé.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.FileSystem.WriteAllBytes**.

Pour	**Consultez**
Écrire dans un fichier binaire	Comment : écrire dans des fichiers binaires dans Visual Basic

Exemple

Cet exemple ajoute le tableau de données `CustomerData` dans le fichier `CollectedData`.

VB

```
My.Computer.FileSystem.WriteAllBytes _
("C:\MyDocuments\CustomerData", CustomerData, True)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : écrire dans des fichiers binaires dans Visual Basic

Référence

My.Computer.FileSystem, objet

WriteAllBytes

Autres ressources

Écriture dans des fichiers en Visual Basic

My.Computer.FileSystem.WriteAllText, méthode

Écrit du texte dans un fichier.

```
' Usage
My.Computer.FileSystem.WriteAllText(file ,text ,append)
My.Computer.FileSystem.WriteAllText(file ,text ,append ,encoding)
' Declaration
Public Sub WriteAllText( _
   ByVal file As String, _
   ByVal text As String, _
   ByVal append As Boolean _
)
' -or-
Public Sub WriteAllText( _
   ByVal file As String, _
   ByVal text As String, _
   ByVal append As Boolean, _
   ByVal encoding As System.Text.Encoding _
)
```

Paramètres

file

String. Fichier dans lequel écrire. Obligatoire.

text

String. Texte à écrire. Requis.

append

Boolean. Si besoin d'ajouter du texte ou de remplacer le texte existant. La valeur par défaut est **False**. Obligatoire.

encoding

Encoding. Quel codage utiliser lors de l'écriture dans un fichier. Obligatoire. La valeur par défaut est UTF-8.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- le chemin d'accès n'est pas valide pour une des raisons suivantes : il s'agit d'une chaîne de longueur nulle ; il ne contient que des espaces blancs ; il contient des caractères non valides ; se termine par une barre oblique finale, ou il s'agit d'un chemin d'accès de périphérique (qui commence par \\.\) (ArgumentException) ;
- le chemin d'accès n'est pas valide, car il a la valeur **Nothing** (ArgumentNullException) ;
- *file* pointe vers un chemin d'accès qui n'existe pas (FileNotFoundException ou DirectoryNotFoundException) ;
- le fichier est utilisé par un autre processus, ou une erreur E/S se produit (IOException) ;
- le chemin d'accès dépasse la longueur maximale définie par le système (PathTooLongException) ;
- un nom de fichier ou de répertoire du chemin d'accès contient un signe deux-points (:) ou n'a pas un format correct (NotSupportedException) ;
- l'utilisateur n'a pas les autorisations nécessaires pour afficher le chemin d'accès (SecurityException).

Si vous exécutez le programme dans un contexte partiellement fiable, le code peut lever une exception en raison de privilèges insuffisants. Pour plus d'informations, consultez Notions fondamentales de la sécurité d'accès du code.

Notes

Lorsque aucun codage n'est spécifié, UTF-8 est utilisé. La marque d'ordre d'octet pour le codage est écrite dans le fichier, sauf si vous spécifiez System.Text.Encoding.Default, qui utilise la page de codes ANSI actuelle du système.

Si le fichier cible n'existe pas, il est créé.

Si *append* a la valeur **False**, le texte existant dans le fichier est remplacé.

Si le codage spécifié ne correspond pas au codage existant du fichier, il est ignoré.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.FileSystem.WriteAllText**.

Pour	Consultez
Écrire du texte dans un fichier	Comment : insérer du texte dans des fichiers dans Visual Basic
Ajouter du texte dans un fichier	Comment : effectuer un ajout à des fichiers texte dans Visual Basic

Exemple

Cet exemple écrit la ligne `"This is new text to be added."` dans le fichier `Test.txt`, ce qui remplace le texte existant dans le fichier.

VB

```
My.Computer.FileSystem.WriteAllText("C:\TestFolder1\test.txt", _
"This is new text to be added.", False)
```

Cet exemple écrit les noms des fichiers du dossier `Documents and Settings` dans `FileList.txt`, ce qui insère un retour chariot entre chacun d'eux pour une meilleure lisibilité.

VB

```
For Each foundFile As String In _
My.Computer.FileSystem.GetFiles("C:\Documents and Settings")
    foundFile = foundFile & vbCrLf
    My.Computer.FileSystem.WriteAllText _
    ("C:\Documents and Settings\FileList.txt", foundFile, True)
Next
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : FileSystemProxy (fournit l'accès à FileSystem)

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem, objet

System.Text.Encoding

Microsoft.VisualBasic.FileIO.FileSystem.WriteAllText

Autres ressources

Écriture dans des fichiers en Visual Basic

My.Computer.FileSystem.SpecialDirectories, objet

Fournit des propriétés pour accéder aux répertoires communément référencés.

Notes

Si le chemin d'accès du répertoire référencé est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire, une exception DirectoryNotFoundException est levée.

Aucune barre oblique inverse (\) n'apparaît à la fin du chemin d'accès.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.FileSystem.SpecialDirectories**.

Pour	**Voir**
Lire le répertoire MyDocuments	Comment : récupérer le contenu du répertoire Mes Documents dans Visual Basic

Exemple

Cet exemple retourne et affiche le `filePath` du répertoire Desktop de l'utilisateur.

VB

```
Dim filePath As String
filePath = My.Computer.FileSystem.SpecialDirectories.Desktop
MsgBox(filePath)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de l'objet My.Computer.FileSystem.SpecialDirectories
My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData, propriété
My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData, propriété
My.Computer.FileSystem.SpecialDirectories.Desktop, propriété
My.Computer.FileSystem.SpecialDirectories.MyDocuments, propriété
My.Computer.FileSystem.SpecialDirectories.MyMusic, propriété
My.Computer.FileSystem.SpecialDirectories.MyPictures, propriété
My.Computer.FileSystem.SpecialDirectories.Programs, propriété
My.Computer.FileSystem.SpecialDirectories.Temp, propriété
My.Computer.FileSystem, objet
Microsoft.VisualBasic.FileIO.SpecialDirectories

Membres de l'objet My.Computer.FileSystem.SpecialDirectories

La propriété My.Computer.FileSystem.SpecialDirectories fournit des propriétés pour accéder aux répertoires communément référencés

Propriétés

AllUsersApplicationData	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire Programs de l'utilisateur.
CurrentUserApplicationData	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire Application Data de l'utilisateur.
Bureau	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire Desktop de l'utilisateur.
MyDocuments	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire MyDocuments de l'utilisateur.
MyMusic	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire MyMusic de l'utilisateur.
MyPictures	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire MyPictures de l'utilisateur.
Programmes	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire Programs de l'utilisateur.
Temp	Chaîne en lecture seule représentant le chemin d'accès qui pointe vers le répertoire Temp de l'utilisateur.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData, propriété

Obtient un chemin d'accès pour stocker les données d'application partagées entre tous les utilisateurs de l'ordinateur.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData
' Declaration
Public ReadOnly Property AllUsersApplicationData As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés.

Si le chemin d'accès n'existe pas, il est créé avec le format suivant : *BasePath**CompanyName**ProductName**Version*.

Un *BasePath* typique de Windows XP est C:\Documents and Settings\All Users\Application Data. *CompanyName*, *ProductName* et *Version* peuvent être obtenus à partir des propriétés My.Application.Info.CompanyName, My.Application.Info.ProductNameet My.Application.Info.Version.

Remarque

Si les noms **Company** et **Product** ne sont pas spécifiés dans la boîte de dialogue **Informations de l'assembly**, la valeur *CompanyName**ProductName* du chemin d'accès est remplacée par le nom de l'assembly. Pour plus d'informations sur la définition des noms des informations de l'assembly, consultez Comment : spécifier des informations d'assembly.

Exemple

Cet exemple affiche le chemin d'accès pour les données des applications partagées dans un MessageBox.

VB

```
MsgBox _
(My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**

Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.AllUsersApplicationData

My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData, propriété

Obtient un chemin d'accès pour stocker les données d'application de l'utilisateur actuel.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData
' Declaration
Public ReadOnly Property CurrentUserApplicationData As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés.

Si le chemin d'accès n'existe pas, il est créé avec le format suivant : *BasePath**CompanyName**ProductName**Version*.

Un *BasePath* typique dans Windows XP est C:\Documents and Settings*nom_utilisateur*\Application Data. *CompanyName*, *ProductName* et *Version* peuvent être obtenus à partir des propriétés My.Application.Info.CompanyName, My.Application.Info.ProductNameet My.Application.Info.Version.

Remarque

Si les noms **Company** et **Product** ne sont pas spécifiés dans la boîte de dialogue **Informations de l'assembly**, la valeur *CompanyName**ProductName* du chemin d'accès est remplacée par le nom de l'assembly. Pour plus d'informations sur la définition des noms des informations de l'assembly, consultez Comment : spécifier des informations d'assembly.

Les données stockées dans ce chemin d'accès font partie d'un profil utilisateur activé pour être itinérant. Un utilisateur itinérant fonctionne sur plusieurs ordinateurs d'un réseau. Le profil utilisateur d'un utilisateur itinérant est conservé sur un serveur du réseau et chargé sur un système lorsque l'utilisateur ouvre une session. Pour qu'un profil utilisateur soit habilité à être itinérant, le système d'exploitation doit prendre en charge les profils itinérants et la fonctionnalité doit être activée.

Exemple

Cet exemple affiche le chemin d'accès pour stocker les données d'application de l'utilisateur actuel dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**

Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante est requise :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.CurrentUserApplicationData

My.Computer.FileSystem.SpecialDirectories.Desktop, propriété

Obtient un nom de chemin d'accès pointant vers le répertoire Bureau.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.Desktop
' Declaration
Public ReadOnly Property Desktop As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

L'objet My.Computer.FileSystem.SpecialDirectories contient les chemins d'accès aux répertoires couramment référencés.

Exemple

Cet exemple affiche le chemin d'accès du répertoire Bureau dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.Desktop)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante est requise :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.Desktop

My.Computer.FileSystem.SpecialDirectories.MyDocuments, propriété

Obtient un nom de chemin d'accès pointant vers le répertoire **Mes documents**.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.MyDocuments
' Declaration
Public ReadOnly Property MyDocuments As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés.

Exemple

Cet exemple affiche le chemin d'accès du répertoire **Mes documents** dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante est requise :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.MyDocuments

My.Computer.FileSystem.SpecialDirectories.MyMusic, propriété

Obtient un nom de chemin d'accès pointant vers le répertoire **Ma musique**.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.MyMusic
' Declaration
Public ReadOnly Property MyMusic As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés.

Exemple

Cet exemple affiche le chemin d'accès du répertoire **Ma musique** dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyMusic)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante est requise :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.MyMusic

My.Computer.FileSystem.SpecialDirectories.MyPictures, propriété

Obtient un nom de chemin d'accès pointant vers le répertoire **Mes images**.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.MyPictures
' Declaration
Public ReadOnly Property MyPictures As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

La classe My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés.

Exemple

Cet exemple affiche le chemin d'accès du répertoire **Mes images** dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyPictures)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante est requise :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.MyPictures

My.Computer.FileSystem.SpecialDirectories.ProgramFiles, propriété

Obtient un nom de chemin d'accès pointant sur le répertoire **Program Files**.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.ProgramFiles
' Declaration
Public ReadOnly Property Programs As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

L'objet My.Computer.FileSystem.SpecialDirectories contient les chemins d'accès aux répertoires couramment référencés. Le répertoire **Program Files** contient les fichiers de programmes de l'ordinateur.

Exemple

Cet exemple affiche le chemin d'accès du répertoire **Program Files** dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.ProgramFiles)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.ProgramFiles

My.Computer.FileSystem.SpecialDirectories.Programs, propriété

Obtient un nom de chemin d'accès pointant vers le répertoire **Programs**.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.Programs
' Declaration
Public ReadOnly Property Programs As String
```

Valeur de retour

String.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés. Le répertoire **Programs** contient les groupes de programmes de l'utilisateur.

Exemple

Cet exemple affiche le chemin d'accès du répertoire **Programs** dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.Programs)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

L'autorisation suivante est requise :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.Programs

My.Computer.FileSystem.SpecialDirectories.Temp, propriété

Obtient un nom de chemin d'accès pointant vers le répertoire temporaire en cours du système.

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.Temp
' Declaration
Public ReadOnly Property Temp As String
```

Valeur de retour

String.

Exceptions

La condition ci-dessous peut générer une exception.

- Le chemin d'accès est vide, généralement parce que le système d'exploitation ne prend pas en charge le répertoire (DirectoryNotFoundException).

Notes

My.Computer.FileSystem.SpecialDirectories, objet contient les chemins d'accès aux répertoires communément référencés.

Exemple

Cet exemple affiche le chemin d'accès pour le répertoire temporaire actuel du système dans un MessageBox.

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.Temp)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : SpecialDirectoriesProxy (fournit l'accès à SpecialDirectories)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Les autorisations suivantes sont requises :

Autorisation	Description
EnvironmentPermission	Contrôle l'accès aux variables du système et de l'environnement utilisateur. Énumération associée : Unrestricted.
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : **Unrestricted**.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.FileSystem.SpecialDirectories, objet

Microsoft.VisualBasic.FileIO.SpecialDirectories.Temp

My.Computer.Info, objet

Fournit des propriétés pour obtenir des informations concernant la mémoire, les assemblys chargés, le nom et le système d'exploitation de l'ordinateur.

Notes

Les propriétés exposées par l'objet **My.Computer.Info** retournent des informations sur l'ordinateur à partir duquel l'application est déployée, comme déterminé au moment de l'exécution. En général, ces données diffèrent de celles disponibles sur l'ordinateur de développement.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.AvailablePhysicalMemory** pour afficher la quantité de mémoire physique disponible sur l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.AvailablePhysicalMemory)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de l'objet My.Computer.Info

Membres de l'objet My.Computer.Info

L'objet My.Computer.Info fournit des propriétés pour obtenir des informations concernant la mémoire, les assemblys chargés, le nom et le système d'exploitation de l'ordinateur.

Propriétés

AvailablePhysicalMemory	Obtient la quantité totale de mémoire physique libre sur l'ordinateur.
AvailableVirtualMemory	Obtient la quantité totale d'espace d'adressage virtuel libre disponible pour l'ordinateur.
InstalledUICulture	Obtient la culture d'interface utilisateur actuelle installée avec le système d'exploitation.
OSFullName	Obtient le nom complet du système d'exploitation.
OSPlatform	Obtient l'identificateur de plate-forme pour le système d'exploitation de l'ordinateur.
OSVersion	Obtient la version du système d'exploitation de l'ordinateur.
TotalPhysicalMemory	Obtient la quantité totale de mémoire physique de l'ordinateur.
TotalVirtualMemory	Obtient la quantité totale d'espace d'adressage virtuel disponible pour l'ordinateur.

Voir aussi

Référence

My.Computer.Info, objet

My.Computer.Info.AvailablePhysicalMemory, propriété

Obtient la quantité de mémoire physique disponible pour l'ordinateur.

```
' Usage
Dim value As ULong = My.Computer.Info.AvailablePhysicalMemory
' Declaration
Public ReadOnly Property AvailablePhysicalMemory As ULong
```

Valeur de retour

ULong contenant le nombre d'octets de mémoire physique disponible pour l'ordinateur.

Exceptions

La condition ci-dessous peut générer une exception :

- l'application ne peut pas obtenir l'état de la mémoire (Win32Exception).

Notes

La propriété **My.Computer.Info.AvailablePhysicalMemory** requiert Windows XP, Windows 2000 Professionnel, Windows Server 2003 ou Windows 2000 Server.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.AvailablePhysicalMemory** pour afficher la quantité de mémoire physique disponible pour l'ordinateur sur lequel le code s'exécute.

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.AvailablePhysicalMemory)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

My.Computer.Info.TotalPhysicalMemory, propriété

My.Computer.Info.AvailableVirtualMemory, propriété

Microsoft.VisualBasic.Devices.ComputerInfo.AvailablePhysicalMemory

My.Computer.Info.AvailableVirtualMemory, propriété

Obtient la quantité totale de l'espace d'adressage virtuel libre de l'ordinateur.

```
' Usage
Dim value As ULong = My.Computer.Info.AvailableVirtualMemory
' Declaration
Public ReadOnly Property AvailableVirtualMemory As ULong
```

Valeur de retour

ULong qui contient le nombre d'octets de l'espace d'adressage virtuel libre de l'ordinateur.

Exceptions

La condition ci-dessous peut générer une exception :

- L'application ne peut pas obtenir l'état de la mémoire (Win32Exception).

Notes

La propriété **My.Computer.Info.AvailableVirtualMemory** requiert Windows XP, Windows 2000 Professionnel, Windows Server 2003 ou Windows 2000 Server.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.AvailableVirtualMemory** pour afficher la quantité de mémoire virtuelle disponible sur l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Computer's available virtual memory: " & _
    My.Computer.Info.AvailableVirtualMemory)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

My.Computer.Info.TotalVirtualMemory, propriété

My.Computer.Info.AvailablePhysicalMemory, propriété

Microsoft.VisualBasic.Devices.ComputerInfo.AvailableVirtualMemory

My.Computer.Info.InstalledUICulture, propriété

Obtient la culture d'interface utilisateur actuelle installée avec le système d'exploitation.

```
' Usage
Dim value As System.Globalization.CultureInfo = My.Computer.Info.InstalledUICulture
' Declaration
Public ReadOnly Property InstalledUICulture As System.Globalization.CultureInfo
```

Valeur de retour

Un objet CultureInfo représente la culture d'interface utilisateur installée sur l'ordinateur.

Notes

La propriété **My.Computer.Info.InstalledUICulture** a le même comportement que la propriété InstalledUICulture.

Dans un système d'exploitation localisé, tel que Windows 2000 Professionnel version japonaise, cette propriété retourne la culture du système d'exploitation.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.InstalledUICulture** pour afficher le nom de la culture d'interface utilisateur du système d'exploitation.

VB

```
MsgBox("Computer's UI culture name: " & _
    My.Computer.Info.InstalledUICulture.DisplayName)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

System.Globalization.CultureInfo
Microsoft.VisualBasic.Devices.ComputerInfo.InstalledUICulture

My.Computer.Info.OSFullName, propriété

Obtient le nom complet du système d'exploitation.

```
' Usage
Dim value As String = My.Computer.Info.OSFullName
' Declaration
Public ReadOnly Property OSFullName As String
```

Valeur de retour

String contenant le nom du système d'exploitation.

Exceptions

La condition ci-dessous peut générer une exception :

- Le code appelant n'a pas une confiance totale (SecurityException).

Notes

Cette propriété retourne des informations détaillées concernant le nom du système d'exploitation si Windows Management Instrumentation (WMI) est installé sur l'ordinateur. Sinon, elle retourne la même chaîne que la propriété **My.Computer.Info.OSPlatform** qui fournit des informations moins détaillées que WMI.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.OSFullName** pour afficher le nom du système d'exploitation de l'ordinateur.

VB

```
MsgBox("Computer's operating system name: " & _
    My.Computer.Info.OSFullName)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.Info, objet

My.Computer.Info.OSPlatform, propriété

Microsoft.VisualBasic.Devices.ComputerInfo.OSFullName

My.Computer.Info.OSPlatform, propriété

Obtient l'identificateur de plate-forme pour le système d'exploitation de l'ordinateur.

```
' Usage
Dim value As String = My.Computer.Info.OSPlatform
' Declaration
Public ReadOnly Property OSPlatform As String
```

Valeur de retour

String qui contient l'identificateur de plate-forme pour le système d'exploitation de l'ordinateur, sélectionné parmi les noms de membres de l'énumération PlatformID.

Exceptions

La condition ci-dessous peut générer une exception :

- L'application ne peut pas obtenir les informations sur la plate-forme du système d'exploitation (ExecutionEngineException).

Notes

La propriété **My.Computer.Info.OSPlatform** fournit des fonctionnalités semblables à la propriété Platform de l'objet retourné par la propriété OSVersion.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.OSPlatform** pour afficher la plate-forme du système d'exploitation de l'ordinateur.

VB

```
MsgBox("Computer's operating system platform: " & _
    My.Computer.Info.OSPlatform)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

Platform

OSVersion

Microsoft.VisualBasic.Devices.ComputerInfo.OSPlatform

My.Computer.Info.OSVersion, propriété

Obtient la version du système d'exploitation de l'ordinateur.

```
' Usage
Dim value As String = My.Computer.Info.OSVersion
' Declaration
Public ReadOnly Property OSVersion As String
```

Valeur de retour

String contenant le numéro de la version actuelle du système d'exploitation.

Exceptions

La condition ci-dessous peut générer une exception :

- L'application ne peut pas obtenir les informations relatives à la version du système d'exploitation (ExecutionEngineException).

Notes

La propriété **My.Computer.Info.OSVersion** met en forme la version sous « major.minor.build.revision ».

La propriété **My.Computer.Info.OSVersion** fournit des fonctionnalités similaires à la propriété Version de l'objet retourné par la propriété OSVersion.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.OSVersion** pour afficher la version du système d'exploitation de l'ordinateur.

VB

```
MsgBox("Computer's operating system version: " & _
    My.Computer.Info.OSVersion)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

Microsoft.VisualBasic.Devices.ComputerInfo.OSVersion

My.Computer.Info.TotalPhysicalMemory, propriété

Obtient la quantité totale de mémoire physique de l'ordinateur.

```
' Usage
Dim value As ULong = My.Computer.Info.TotalPhysicalMemory
' Declaration
Public ReadOnly Property TotalPhysicalMemory As ULong
```

Valeur de retour

ULong qui contient le nombre d'octets de mémoire physique de l'ordinateur.

Exceptions

La condition ci-dessous peut générer une exception :

- L'application ne peut pas obtenir l'état de la mémoire (Win32Exception).

Notes

La propriété **My.Computer.Info.TotalPhysicalMemory** requiert Windows XP, Windows 2000 Professionnel, Windows Server 2003 ou Windows 2000 Server.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.TotalPhysicalMemory** pour afficher la quantité totale de mémoire physique de l'ordinateur sur lequel s'exécute le code.

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.TotalPhysicalMemory)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

My.Computer.Info.AvailablePhysicalMemory, propriété

My.Computer.Info.TotalVirtualMemory, propriété

Microsoft.VisualBasic.Devices.ComputerInfo.TotalPhysicalMemory

My.Computer.Info.TotalVirtualMemory, propriété

Obtient la quantité d'espace disponible des adresses virtuelles de l'ordinateur.

```
' Usage
Dim value As ULong = My.Computer.Info.TotalVirtualMemory
' Declaration
Public ReadOnly Property TotalVirtualMemory As ULong
```

Valeur de retour

ULong contenant le nombre d'octets d'espace disponible des adresses virtuelles de l'ordinateur.

Exceptions

La condition ci-dessous peut générer une exception :

- l'application ne peut pas obtenir l'état de la mémoire (Win32Exception).

Notes

La propriété **My.Computer.Info.TotalVirtualMemory** requiert Windows XP, Windows 2000 Professionnel, Windows Server 2003 ou Windows 2000 Server.

Exemple

Cet exemple utilise la propriété **My.Computer.Info.TotalVirtualMemory** pour afficher la quantité totale de mémoire virtuelle de l'ordinateur sur lequel le code s'exécute.

VB

```
MsgBox("Computer's available virtual memory: " & _
    My.Computer.Info.TotalVirtualMemory)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : ComputerInfo

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Info, objet

My.Computer.Info.AvailableVirtualMemory, propriété

My.Computer.Info.TotalPhysicalMemory, propriété

Microsoft.VisualBasic.Devices.ComputerInfo.TotalVirtualMemory

My.Computer.Keyboard, objet

Fournit des propriétés pour accéder à l'état actuel du clavier, par exemple pour savoir quelles touches sont actuellement utilisées, et fournit une méthode pour envoyer des séquences de touches à la fenêtre active.

Notes

L'objet **My.Computer.Keyboard** fournit une interface au clavier de l'ordinateur. Les propriétés de **My.Computer.Keyboard** fournissent des informations sur l'état de plusieurs touches spéciales. La méthode **My.Computer.Keyboard.SendKeys** vous permet d'envoyer des touches dans la fenêtre active comme si elles ont été tapées sur le clavier.

Tâche

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Keyboard**.

Pour	Consultez
Vérifier si la touche VERR MAJ est activée	Comment : déterminer si la touche Verr. Maj. est activée dans Visual Basic
Envoyer des séquences de touches à une application	Comment : démarrer une application et envoyer des séquences de touches (Visual Basic)

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.CtrlKeyDown** pour déterminer si la touche CTRL de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.CtrlKeyDown Then
    MsgBox("CTRL key down")
Else
    MsgBox("CTRL key up")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Voir aussi

Référence

Membres de l'objet My.Computer.Keyboard
My.Computer, objet
Microsoft.VisualBasic.Devices.Keyboard

Concepts

Accès au clavier

Membres de l'objet My.Computer.Keyboard

L'objet My.Computer.Keyboard fournit des propriétés pour accéder à l'état actuel du clavier, par exemple pour savoir quelles touches sont actuellement utilisées, et fournit une méthode pour envoyer des séquences de touches à la fenêtre active.

Propriétés

AltKeyDown	Obtient **Boolean** qui indique si la touche ALT est activée. Cette propriété est disponible uniquement pour les applications non-serveurs.
CapsLock	Obtient **Boolean** qui indique si la touche VERR MAJ est activée. Cette propriété est disponible uniquement pour les applications non-serveurs.
CtrlKeyDown	Obtient **Boolean** qui indique si la touche CTRL est activée. Cette propriété est disponible uniquement pour les applications non-serveurs.
NumLock	Obtient **Boolean** qui indique si la touche VERR NUM est activée. Cette propriété est disponible uniquement pour les applications non-serveurs.
ScrollLock	Obtient **Boolean** qui indique si la touche ARRÊT DÉFIL est activée. Cette propriété est disponible uniquement pour les applications non-serveurs.
ShiftKeyDown	Obtient **Boolean** qui indique si la touche MAJ est activée. Cette propriété est disponible uniquement pour les applications non-serveurs.

Méthodes

SendKeys	Envoie une ou plusieurs séquences de touches dans la fenêtre active, comme si elles sont tapées sur le clavier. Cette méthode est disponible uniquement pour les applications non-serveurs.

Voir aussi

Référence

My.Computer.Keyboard, objet

My.Computer.Keyboard.AltKeyDown, propriété

Obtient **Boolean** qui indique si la touche ALT est activée.

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.AltKeyDown
' Declaration
Public ReadOnly Property AltKeyDown As Boolean
```

Valeur de retour

Valeur **Boolean** : **True** si la touche ALT est activée ; sinon **False**.

Notes

La propriété **My.Computer.Keyboard.AltKeyDown** fournit des fonctionnalités semblables à la propriété ModifierKeys.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.AltKeyDown** pour déterminer si la touche ALT de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.AltKeyDown Then
    MsgBox("ALT key down")
Else
    MsgBox("ALT key up")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Keyboard, objet

ModifierKeys
Microsoft.VisualBasic.Devices.Keyboard.AltKeyDown

My.Computer.Keyboard.CapsLock, propriété

Obtient un **Boolean** indiquant si la touche VERR. MAJ est activée.

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.CapsLock
' Declaration
Public ReadOnly Property CapsLock As Boolean
```

Valeur de retour

Valeur **Boolean** : **True** si la touche VERR. MAJ est activée ; sinon **False**.

Notes

Seul l'état de la touche VERR. MAJ est retourné.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.CapsLock** pour déterminer si la touche VERR. MAJ de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.CapsLock Then
    MsgBox("CAPS LOCK on")
Else
    MsgBox("CAPS LOCK off")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Keyboard, objet

Microsoft.VisualBasic.Devices.Keyboard.CapsLock

My.Computer.Keyboard.CtrlKeyDown, propriété

Obtient **Boolean** qui indique si la touche CTRL est activée.

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.CtrlKeyDown
' Declaration
Public ReadOnly Property CtrlKeyDown As Boolean
```

Valeur de retour

Valeur **Boolean** : **True** si la touche CTRL est activée ; sinon **False**.

Notes

La propriété **My.Computer.Keyboard.CtrlKeyDown** fournit des fonctionnalités semblables à la propriété ModifierKeys.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.CtrlKeyDown** pour déterminer si la touche CTRL de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.CtrlKeyDown Then
    MsgBox("CTRL key down")
Else
    MsgBox("CTRL key up")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Keyboard, objet

ModifierKeys
Microsoft.VisualBasic.Devices.Keyboard.CtrlKeyDown

My.Computer.Keyboard.NumLock, propriété

Obtient **Boolean** qui indique si la touche VERR NUM est activée.

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.NumLock
' Declaration
Public ReadOnly Property NumLock As Boolean
```

Valeur de retour

Valeur **Boolean**. **True** si la touche VERR NUM est activée ; sinon **False**.

Notes

Seul l'état de la touche VERR NUM est retourné.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.NumLock** pour déterminer si la touche VERR NUM de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.NumLock Then
    MsgBox("NUM LOCK on")
Else
    MsgBox("NUM LOCK off")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Keyboard, objet

Microsoft.VisualBasic.Devices.Keyboard.NumLock

My.Computer.Keyboard.ScrollLock, propriété

Obtient **Boolean** qui indique si la touche ARRÊT DÉFIL est activée.

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.ScrollLock
' Declaration
Public ReadOnly Property ScrollLock As Boolean
```

Valeur de retour

Valeur **Boolean**. **True** si la touche ARRÊT DÉFIL est activée ; sinon **False**.

Notes

Seul l'état de la touche ARRÊT DÉFIL est retourné.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.ScrollLock** pour déterminer si la touche ARRÊT DÉFIL de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.ScrollLock Then
    MsgBox("SCROLL LOCK on")
Else
    MsgBox("SCROLL LOCK off")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Keyboard, objet

Microsoft.VisualBasic.Devices.Keyboard.ScrollLock

My.Computer.Keyboard.SendKeys, méthode

Envoie une ou plusieurs séquences de touches dans la fenêtre active, comme si elles sont tapées sur le clavier.

```
' Usage
My.Computer.Keyboard.SendKeys(keys)
My.Computer.Keyboard.SendKeys(keys ,wait)
' Declaration
Public Sub SendKeys( _
   ByVal keys As String _
)
' -or-
Public Sub SendKeys( _
   ByVal keys As String, _
   ByVal wait As Boolean _
)
```

Paramètres

keys

String qui définit les touches à envoyer.

wait

Facultatif. **Boolean** qui spécifie s'il faut attendre la fin du traitement des séquences de touches avant que l'application continue. **True** par défaut.

Exceptions

La condition ci-dessous peut générer une exception :

- Il existe une situation d'un niveau de confiance partiel dans laquelle l'utilisateur ne dispose des autorisations nécessaires (SecurityException).

Notes

La méthode **My.Computer.Keyboard.SendKeys** fournit des fonctionnalités semblables aux méthodes Send et SendWait.

L'argument *wait* est utile si l'autre application doit arriver à son terme avant que votre application puisse continuer.

Remarque

Étant donné qu'il n'existe pas de méthode gérée pour activer une autre application, vous pouvez utiliser cette classe dans l'application actuelle, sélectionner la fenêtre dans laquelle envoyer les touches ou utiliser les méthodes API Windows, telles que **FindWindow** et **SetForegroundWindow**, pour forcer le focus sur d'autres applications. Pour plus d'informations, consultez Procédure pas à pas : appel des API Windows.

L'argument *keys* peut spécifier une seule touche ou une touche combinée avec ALT, CTRL ou MAJ (ou toute combinaison de ces touches). Chaque touche est représentée par un ou plusieurs caractères, par exemple `a` pour le caractère "a" ou `{ENTER}` pour la touche ENTRÉE.

Pour combiner une touche avec MAJ, précédez le code de la touche de `+` (signe plus). Pour combiner une touche avec CTRL, précédez le code de la touche de `^` (signe insertion). Pour combiner une touche avec ALT, précédez le code de la touche de `%` (signe pourcentage). Pour spécifier la répétition des touches, utilisez la forme `{`*key number*`}`. Vous devez insérer un espace entre *key* et *number*. Par exemple, `{LEFT 42}` signifie « appuyez sur la touche GAUCHE 42 fois » ; `{h 10}` signifie « appuyez sur 'h' 10 fois ».

Le tableau suivant répertorie les codes qui peuvent être utilisés pour spécifier des caractères qui ne s'affichent pas lorsque vous appuyez sur la touche correspondante (par exemple, ENTRÉE ou TAB).

Touche	Code
RET.ARR	{RET. ARR} ou {RA}

ATTN	{ATTN}
VERR. MAJ	{VERR MAJ}
EFFACER	{EFFACER}
SUPPR	{SUPPR}
BAS	{BAS}
FIN	{FIN}
ENTRÉE (pavé numérique)	{ENTRÉE}
ENTRÉE	~
ÉCHAP	{ÉCHAP}
AIDE	{AIDE}
DÉBUT	{DÉBUT}
INS	{INSER}
GAUCHE	{GAUCHE}
VERR. NUM	{VERR. NUM}
PG.SUIV	{PG. SUIV}
PG.PRÉC	{PG. PRÉC}
RETOUR	{RETOUR}
DROITE	{DROITE}
ARRÊT DÉFIL	{ARRÊT DÉFIL}
TAB	{TAB}
HAUT	{HAUT}
F1 à F15	{F1} à {F15}

Exemple

Cet exemple utilise la méthode **My.Computer.Keyboard.SendKeys** pour envoyer des séquences de touches à une application externe, l'application Calculatrice, démarrée par la fonction **Shell**.

VB

```
Dim ProcID As Integer
' Start the Calculator application, and store the process id.
ProcID = Shell("CALC.EXE", AppWinStyle.NormalFocus)
' Activate the Calculator application.
AppActivate(ProcID)
```

```
' Send the keystrokes to the Calculator application.
My.Computer.Keyboard.SendKeys("22", True)
My.Computer.Keyboard.SendKeys("*", True)
My.Computer.Keyboard.SendKeys("44", True)
My.Computer.Keyboard.SendKeys("=", True)
' The result is 22 * 44 = 968.
```

Une exception ArgumentException est levée si une application contenant l'identificateur de processus demandé est introuvable.

L'appel à la fonction **Shell** requiert une confiance totale (classe **SecurityException**).

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : AllWindows.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Procédure pas à pas : appel des API Windows

Référence

My.Computer.Keyboard, objet

Send

SendWait

Microsoft.VisualBasic.Devices.Keyboard.SendKeys(System.String,System.Boolean)

My.Computer.Keyboard.ShiftKeyDown, propriété

Obtient **Boolean** qui indique si la touche MAJ est activée.

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.ShiftKeyDown
' Declaration
Public ReadOnly Property ShiftKeyDown As Boolean
```

Valeur de retour

Valeur **Boolean** : **True** si la touche MAJ est activée ; sinon **False**.

Notes

La propriété **My.Computer.Keyboard.ShiftKeyDown** fournit des fonctionnalités semblables à la propriété ModifierKeys.

Exemple

Cet exemple utilise la propriété **My.Computer.Keyboard.ShiftKeyDown** pour déterminer si l'une des touches MAJ de l'ordinateur est activée.

VB

```
If My.Computer.Keyboard.ShiftKeyDown Then
    MsgBox("SHIFT key down")
Else
    MsgBox("SHIFT key up")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Keyboard

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Keyboard, objet

ModifierKeys
Microsoft.VisualBasic.Devices.Keyboard.ShiftKeyDown

My.Computer.Mouse, objet

Fournit les propriétés permettant d'obtenir des informations relatives au format et à la configuration de la souris installée sur l'ordinateur local.

Notes

L'objet **My.Computer.Mouse** permet de rechercher des informations sur la souris de l'ordinateur, notamment sur la permutation ou non des boutons et les détails concernant la roulette.

Exemple

Cet exemple utilise les propriétés **My.Computer.Mouse.WheelExists** et **My.Computer.Mouse.WheelScrollLines** pour déterminer si la souris a une roulette de défilement et le nombre d'éléments qu'elle fait défiler lorsque vous la faites pivoter.

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Mouse

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Voir aussi

Référence

Membres de l'objet My.Computer.Mouse
My.Computer, objet
Microsoft.VisualBasic.Devices.Mouse

Membres de l'objet My.Computer.Mouse

L'objet My.Computer.Mouse fournit des propriétés pour accéder aux informations sur le format et la configuration de la souris installée sur l'ordinateur local.

Propriétés

Propriété	Description
ButtonsSwapped	Obtient **Boolean** qui indique si les fonctions des boutons gauche et droit de la souris ont été permutées. Cette propriété est disponible uniquement pour les applications non-serveurs.
WheelExists	Obtient **Boolean** qui indique si la souris possède une roulette de défilement. Cette propriété est disponible uniquement pour les applications non-serveurs.
WheelScrollLines	Obtient un nombre qui indique la longueur du défilement à l'écran lorsque la roulette de la souris est tourné e d'un cran. Cette propriété est disponible uniquement pour les applications non-serveurs.

Voir aussi

Référence

My.Computer.Mouse, objet

My.Computer.Mouse.ButtonsSwapped, propriété

Obtient **Boolean** qui indique si les fonctions des boutons gauche et droit de la souris ont été permutées.

```
' Usage
Dim value As Boolean = My.Computer.Mouse.ButtonsSwapped
' Declaration
Public ReadOnly Property ButtonsSwapped As Boolean
```

Valeur de retour

Boolean qui a la valeur **True** si les fonctions des boutons gauche et droit de la souris ont été permutées ; sinon, **False**.

Exceptions

La condition ci-dessous peut générer une exception :

- Aucune souris n'est installée sur l'ordinateur (InvalidOperationException).

Notes

La propriété **My.Computer.Mouse.ButtonsSwapped** :

- fournit des fonctionnalités semblables à la propriété MouseButtonsSwapped ;
- fournit des informations sur l'ordinateur qui exécute le code.

Exemple

Cet exemple utilise la propriété **My.Computer.Mouse.ButtonsSwapped** pour déterminer si les fonctions des boutons gauche et droit de la souris ont été permutées.

VB

```
If My.Computer.Mouse.ButtonsSwapped Then
    MsgBox("Functionality of the mouse buttons is swapped.")
Else
    MsgBox("Default functionality of the mouse buttons.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Mouse

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**

Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Mouse, objet

Microsoft.VisualBasic.Devices.Mouse.ButtonsSwapped

My.Computer.Mouse.WheelExists, propriété

Obtient **Boolean** qui indique si la souris possède une roulette de défilement.

```
' Usage
Dim value As Boolean = My.Computer.Mouse.WheelExists
' Declaration
Public ReadOnly Property WheelExists As Boolean
```

Valeur de retour

Boolean qui a la valeur **True** si la souris possède une roulette de défilement ; sinon **False**.

Exceptions

La condition ci-dessous peut générer une exception :

- Aucune souris n'est installée sur l'ordinateur (InvalidOperationException).

Notes

La propriété **My.Computer.Mouse.WheelExists** fournit des fonctionnalités semblables à la propriété MouseWheelPresent.

Vous devez vérifier la valeur de la propriété **My.Computer.Mouse.WheelExists** vant d'essayer d'accéder à la propriété My.Computer.Mouse.WheelScrollLines, qui lève une exception si la souris ne possède pas de roulette de défilement.

La propriété **My.Computer.Mouse.WheelExists** fournit des informations sur l'ordinateur qui exécute le code.

Exemple

Cet exemple utilise la propriété **My.Computer.Mouse.WheelExists** pour déterminer si la souris a une roulette de défilement et la longueur de défilement lorsque vous faites pivoter la roulette.

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Mouse

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**

Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Mouse, objet

My.Computer.Mouse.WheelScrollLines, propriété

Microsoft.VisualBasic.Devices.Mouse.WheelExists

My.Computer.Mouse.WheelScrollLines, propriété

Obtient un nombre qui indique la longueur du défilement à l'écran lorsque la roulette de la souris est tournée d'un cran.

```
' Usage
Dim value As Integer = My.Computer.Mouse.WheelScrollLines
' Declaration
Public ReadOnly Property WheelScrollLines As Integer
```

Valeur de retour

Integer qui indique la longueur du défilement à l'écran lorsque la roulette de la souris est tournée d'un cran. Une valeur positive indique un défilement du nombre de lignes spécifié, et une valeur négative indique un défilement d'un écran à la fois.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Aucune souris n'est installée sur l'ordinateur (InvalidOperationException).
- La souris n'a pas de roulette de défilement (**InvalidOperationException**).

Notes

Vous pouvez utiliser cette propriété pour déterminer le nombre de lignes à défiler vers le haut ou vers le bas dans un contrôle multiligne qui contient une barre de défilement.

La propriété **My.Computer.Mouse.WheelScrollLines** fournit des fonctionnalités semblables à la propriété MouseWheelScrollLines.

Étant donné que cette propriété lève une exception si la souris ne possède pas de roulette de défilement, vous devez vous assurer qu'elle en possède une en vérifiant la propriété **My.Computer.Mouse.WheelExists**.

Cette propriété fournit des informations sur l'ordinateur qui exécute le code.

Exemple

Cet exemple détermine si la souris a une roulette de défilement et utilise la propriété **My.Computer.Mouse.WheelExists** pour déterminer la longueur du défilement à l'écran lorsque vous faites pivoter la roulette de la souris.

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Mouse

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Mouse, objet

My.Computer.Mouse.WheelExists, propriété

Microsoft.VisualBasic.Devices.Mouse.WheelScrollLines

My.Computer.Network, objet

Fournit une propriété, un événement et des méthodes pour interagir avec le réseau auquel l'ordinateur est connecté.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Network**.

Pour	Consultez
Transférer un fichier	Comment : transférer un fichier dans Visual Basic
Télécharger un fichier	Comment : télécharger un fichier dans Visual Basic
Vérifier l'état de la connexion	Comment : vérifier l'état de la connexion dans Visual Basic
Déterminer si un ordinateur distant est disponible	Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Exemple

Cet exemple transfère le fichier `Order.txt` dans http://www.cohowinery.com/uploads.

VB

```
My.Computer.Network.UploadFile ( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/uploads.aspx")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Network

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de l'objet My.Computer.Network
My.Computer, objet
My.Computer.Network.NetworkAvailabilityChanged, événement
Microsoft.VisualBasic.Devices.Network

Membres de l'objet My.Computer.Network

Fournit une propriété, un événement et des méthodes pour accéder aux informations sur le réseau et pour transférer et télécharger des fichiers.

Propriétés

Propriété	Description
IsAvailable	Indique si le réseau est disponible.

Méthodes

Méthode	Description
DownloadFile	Télécharge un fichier à partir d'un emplacement distant.
Ping	Exécute une commande ping sur un ordinateur distant.
UploadFile	Transfère un fichier à partir d'un emplacement distant.

Événements

Événement	Description
NetworkAvailabilityChanged Event	Se produit lorsque l'état de la connexion réseau change.

Voir aussi

Référence

My.Computer.Network, objet

Autres ressources

Exécution d'opérations de réseau

My.Computer.Network.DownloadFile, méthode

Télécharge le fichier distant spécifié et l'enregistre à l'emplacement spécifié.

```
' Usage
My.Computer.Network.DownloadFile(address ,destinationFileName)
My.Computer.Network.DownloadFile(address ,destinationFileName)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite ,onUserCancel)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite ,onUserCancel)
My.Computer.Network.DownloadFile(address ,destinationFileName ,networkCredentials ,showUI ,
connectionTimeout ,overwrite)
My.Computer.Network.DownloadFile(address ,destinationFileName ,networkCredentials ,showUI ,
connectionTimeout ,overwrite ,onUserCancel)
' Declaration
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As System.Uri, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As System.Uri, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal overwrite As Boolean, _
```

```
    ByVal onUserCancel As UICancelOption _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal userName As String, _
    ByVal password As String, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal userName As String, _
    ByVal password As String, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean, _
    ByVal onUserCancel As UICancelOption _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal networkCredentials As System.Net.ICredentials, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal networkCredentials As System.Net.ICredentials, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean, _
    ByVal onUserCancel As UICancelOption _
)
```

Paramètres

address

String ou Uri. Chemin d'accès au fichier à télécharger, y compris nom de fichier et adresse d'hôte. Obligatoire.

destinationFileName

String. Chemin d'accès et nom du fichier téléchargé. Obligatoire.

userName

String. Nom d'utilisateur pour authentification. La valeur par défaut est une chaîne vide, "".

password

String. Mot de passe pour authentification. La valeur par défaut est une chaîne vide, "".

showUI

Boolean. Spécifie si besoin d'afficher l'avancement de l'opération. La valeur par défaut est **False**.

connectionTimeout

Int32. Intervalle de délai en millisecondes. La valeur par défaut est de 100 secondes.

overwrite

Boolean. Spécifie si besoin de remplacer des fichiers existants. La valeur par défaut est **False**.

onUserCancel

UICancelOption. Spécifie le comportement lorsque l'utilisateur clique sur **Annuler** ou sur **Non** dans la boîte de dialogue qui s'affiche lorsque *ShowUI* a la valeur **True**. La valeur par défaut est ThrowException.

networkCredentials

ICredentials. Informations d'identification à fournir.

Exceptions

Les conditions ci-dessous peuvent lever une exception :

- Le nom du lecteur n'est pas valide (ArgumentException).
- *destinationFileName* se termine par une barre oblique finale (**ArgumentException**).
- *overwrite* a la valeur **False** et le fichier de destination existe déjà (IOException).
- Le serveur ne répond pas dans l'intervalle *connectionTimeout* spécifié (TimeoutException).
- L'authentification a échoué (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (**SecurityException**).
- La demande est refusée par le site Web (WebException).

Notes

Si *showUI* a la valeur **True**, une boîte de dialogue s'affiche en indiquant l'avancement de l'opération ; elle comporte un bouton **Annuler** que l'utilisateur peut utiliser pour annuler l'opération. Cette boîte de dialogue n'étant pas modale, elle ne bloque pas les entrées d'utilisateur dans les autres fenêtres du programme.

Si le serveur ne répond pas dans l'intervalle *connectionTimeout* spécifié, l'opération est annulée et une exception est levée.

DownloadFile génère des informations de traçage lorsque vous activez le traçage réseau dans votre application. Pour plus d'informations, consultez Activation du traçage réseau.

Remarque
La méthode **DownloadFile** n'envoie pas d'en-têtes HTTP facultatifs. Certains serveurs peuvent retourner 500 (Erreur interne du serveur) en cas d'absence de l'en-tête d'agent utilisateur (user agent) facultatif. Pour envoyer des en-têtes facultatifs, vous devez créer une demande à l'aide de la classe WebClient. Pour plus d'informations, consultez Opérations de réseau dans le .NET Framework avec Visual Basic.

Remarque de sécurité
Le protocole FTP envoie les informations, y compris les mots de passe, en texte brut, et ne doit pas être utilisé pour transmettre des informations sensibles.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.Network.DownloadFile**.

Pour	**Consultez**
Télécharger un fichier	Comment : télécharger un fichier dans Visual Basic

Exemple

Cet exemple télécharge le fichier `WineList.txt` à partir de `http://www.cohowinery.com/downloads` et l'enregistre dans `C:\Documents and Settings\All Users\Documents`.

VB

```
My.Computer.Network.DownloadFile _
("http://www.cohowinery.com/downloads/WineList.txt", _
"C:\Documents and Settings\All Users\Documents\WineList.txt")
```

Cet exemple télécharge le fichier `WineList.txt` à partir de `http://www.cohowinery.com/downloads` et l'enregistre dans

`C:\Documents and Settings\All Users\Documents`, en spécifiant un intervalle de délai d'attente de 500 millisecondes.

VB

```
My.Computer.Network.DownloadFile _
("http://www.cohowinery.com/downloads/", _
"C:\Documents and Settings\All Users\Documents\WineList.txt", _
"", "", False, 500, True)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Network

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : AllWindows.
WebPermission	Contrôle les droits d'accès aux ressources Internet HTTP. Énumération associée : **Unrestricted**.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : télécharger un fichier dans Visual Basic
Comment : déterminer si un ordinateur distant est disponible dans Visual Basic
Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.Network, objet
System.Uri
System.Net.ICredentials
Microsoft.VisualBasic.Devices.Network.DownloadFile

My.Computer.Network.IsAvailable, propriété

Indique si un ordinateur est connecté à un réseau.

```
' Usage
Dim value As Boolean = My.Computer.Network.IsAvailable
' Declaration
Public ReadOnly Property IsAvailable As Boolean
```

Valeur de retour

Boolean.

Notes

IsAvailable retourne toujours **False** en cas d'appel à partir d'une application **Click Once** ou si l'utilisateur ne dispose pas de NetworkInformationPermission.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.Computer.Network.IsAvailable**.

Pour	Consultez
Vérifier si un ordinateur est connecté à un réseau	Comment : vérifier l'état de la connexion dans Visual Basic

Exemple

Cet exemple vérifie l'état de la propriété et le notifie.

VB

```
If My.Computer.Network.IsAvailable = True Then
    MsgBox("Computer is connected.")
Else
    MsgBox("Computer is not connected.")
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Network

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
WebPermission	Contrôle les droits d'accès aux ressources Internet HTTP. Énumération associée : **Unrestricted**.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Référence

My.Computer.Network, objet

Microsoft.VisualBasic.Devices.Network.IsAvailable

My.Computer.Network.Ping, méthode

Exécute une commande ping sur le serveur spécifié.

```
' Usage
Dim value As Boolean = My.Computer.Network.Ping(hostNameOrAddress)
Dim value As Boolean = My.Computer.Network.Ping(address)
Dim value As Boolean = My.Computer.Network.Ping(hostNameOrAddress ,timeout)
Dim value As Boolean = My.Computer.Network.Ping(address ,timeout)
' Declaration
Public Function Ping( _
   ByVal hostNameOrAddress As String _
) As Boolean
' -or-
Public Function Ping( _
   ByVal address As System.Uri _
) As Boolean
' -or-
Public Function Ping( _
   ByVal hostNameOrAddress As String, _
   ByVal timeout As Integer _
) As Boolean
' -or-
Public Function Ping( _
   ByVal address As System.Uri, _
   ByVal timeout As Integer _
) As Boolean
```

Paramètres

hostNameOrAddress

String. URL, nom d'ordinateur ou numéro IP du serveur sur lequel exécuter une commande ping. Obligatoire.

address

Uri. URI du serveur sur lequel exécuter une commande ping. Obligatoire.

timeout

Int32. Seuil horaire en millisecondes pour contacter la destination. La valeur par défaut est 500. Obligatoire.

Valeur de retour

Boolean qui indique si l'opération a réussi.

Exceptions

Les conditions ci-dessous peuvent lever une exception :

- Aucune connexion réseau n'est disponible (InvalidOperationException).
- L'URL n'est pas valide (PingException).

Notes

La méthode **Ping** ne constitue pas une méthode infaillible pour déterminer la disponibilité d'un ordinateur distant : le port ping sur l'ordinateur cible peut être désactivé ou la demande ping peut être bloquée par un pare-feu ou un routeur.

L'adresse passée à la méthode **Ping** doit prendre en charge la résolution DNS et ne peut pas être précédée de « http:// ».

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Network.Ping**.

Pour	**Consultez**

Déterminer si un ordinateur ou un hôte distant est disponible	Comment : déterminer si un ordinateur distant est disponible dans Visual Basic

Exemple

Cet exemple indique si une commande ping peut être exécutée sur le serveur en déterminant si la méthode **Ping** retourne **True**.

VB

```
If My.Computer.Network.Ping("198.01.01.01") Then
   MsgBox("Server pinged successfully.")
Else
   MsgBox("Ping request timed out.")
End If
```

Remplacez "198.01.01.01" par l'adresse IP, l'URL ou le nom d'ordinateur du serveur sur lequel exécuter une commande ping.

Cet exemple indique si une commande ping peut être exécutée sur le serveur en déterminant si la méthode **Ping** a retourné **True**, et spécifie un intervalle de délai d'attente de 1000 millisecondes.

VB

```
If My.Computer.Network.Ping("www.cohowinery.com",1000) Then
   MsgBox("Server pinged successfully.")
Else
   MsgBox("Ping request timed out.")
End If
```

Remplacez "www.cohowinery.com" par l'adresse IP, l'URL ou le nom d'ordinateur du serveur sur lequel exécuter une commande ping.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Network

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	Description

FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : ControlPrincipal.
WebPermission	Contrôle les droits d'accès aux ressources Internet HTTP. Énumération associée : **Unrestricted**.
SocketPermission	Contrôle les droits de création ou d'acception de connexions sur une adresse de transport. Énumération associée : **Unrestricted**.
PerformanceCounterPermission	Contrôle l'accès à un composant de compteur de performance Windows NT. Énumération associée : **Unrestricted**.
NetworkInformationPermission	Contrôle l'accès aux informations réseau et aux statistiques de trafic pour l'ordinateur local. Énumération associée : Ping.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

My.Computer.Network, objet

System.Uri

Microsoft.VisualBasic.Devices.Network.Ping

My.Computer.Network.NetworkAvailabilityChanged, événement

Se produit lorsque la disponibilité du réseau change.

```
' Usage
Public Sub MyComputerNetwork_NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As Devices.NetworkAvailableEventArgs _
)
End Sub
Sub Handle_NetworkAvailabilityChanged()
   AddHandler My.Computer.Network.NetworkAvailabilityChanged, _
      AddressOf MyComputerNetwork_NetworkAvailabilityChanged
End Sub
' Declaration
Public Event NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As Devices.NetworkAvailableEventArgs _
)
```

Paramètres

sender

Obligatoire. Objet **Object** qui a déclenché l'événement.

e

Obligatoire. Objet NetworkAvailableEventArgs qui contient des informations sur la disponibilité du réseau.

Notes

Une application déclenche l'événement **NetworkAvailabilityChanged** chaque fois que la disponibilité du réseau change. Vous pouvez utiliser la propriété IsNetworkAvailable du paramètre *e* pour obtenir le nouvel état de la connexion réseau. Pour obtenir l'état actuel de la connexion réseau, utilisez la propriété My.Computer.Network.IsAvailable.

Dans une application Windows Forms, cet événement est déclenché sur le thread principal de l'application avec les autres événements de l'interface utilisateur. Ainsi, le gestionnaire d'événements peut directement accéder à l'interface utilisateur de l'application. Toutefois, si l'application est occupée par la gestion d'un autre événement de l'interface utilisateur lorsque cet événement est déclenché, ce dernier ne pourra pas être traité tant que l'autre gestionnaire d'événements n'a pas terminé ou appelé la méthode My.Application.DoEvents.

Vous ne pouvez pas utiliser l'instruction **Handles** pour gérer l'événement **NetworkAvailabilityChanged** ; vous devez utiliser l'instruction **AddHandler**.

Dans les applications Windows Forms, l'objet **My.Application** expose un événement **NetworkAvailabilityChanged** qui fournit les mêmes fonctionnalités que cet événement, mais vous pouvez le gérer avec l'instruction **Handles**. Pour plus d'informations, consultez My.Application.NetworkAvailabilityChanged, événement.

Remarque

De nombreux concentrateurs réseau fournissent une connexion réseau, même si le concentrateur est déconnecté d'un réseau de plus grande taille. Par conséquent, pour les connexions câblées, cet événement indique une modification de la connexion entre l'ordinateur et un concentrateur.

Remarque

L'événement **NetworkAvailabilityChanged** n'est pas déclenché par les applications qui s'exécutent sur Windows 95 et sur Windows 98 ni par les applications exécutées sur Windows 2000 par un utilisateur qui n'est pas administrateur. Si votre appl ication peut s'exécuter sur ces plates-formes, utilisez la propriété My.Computer.Network.IsAvailable pour vérifier la disponibil ité du réseau.

Exemple

Cet exemple utilise l'événement **My.Computer.Network.NetworkAvailabilityChanged** pour mettre à jour l'interface utilisateur d'un formulaire ou d'un contrôle.

VB

```
Private Sub DisplayAvailability(ByVal available As Boolean)
    Label1.Text = available.ToString
End Sub

Private Sub MyComputerNetwork_NetworkAvailabilityChanged( _
    ByVal sender As Object, _
    ByVal e As Devices.NetworkAvailableEventArgs)

    DisplayAvailability(e.IsNetworkAvailable)
End Sub

Private Sub Handle_NetworkAvailabilityChanged()
    AddHandler My.Computer.Network.NetworkAvailabilityChanged, _
       AddressOf MyComputerNetwork_NetworkAvailabilityChanged
    DisplayAvailability(My.Computer.Network.IsAvailable)
End Sub
```

Ce code doit figurer dans un formulaire ou dans un contrôle dont l'Label est nommée `Label1`. Vous devez appeler la méthode `Handle_NetworkAvailabilityChanged` pour initialiser l'étiquette et raccorder le gestionnaire d'événements.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Network

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

My.Computer.Network, objet
My.Application.NetworkAvailabilityChanged, événement
Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs

My.Computer.Network.UploadFile, méthode

Envoie le fichier spécifié à l'adresse de l'hôte indiquée.

```
' Usage
My.Computer.Network.UploadFile(sourceFileName ,address)
My.Computer.Network.UploadFile(sourceFileName ,address)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout ,onUserCancel)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout ,onUserCancel)
My.Computer.Network.UploadFile(sourceFileName ,address ,networkCredentials ,showUI ,connect
ionTimeout)
My.Computer.Network.UploadFile(sourceFileName ,address ,networkCredentials ,showUI ,connect
ionTimeout ,onUserCancel)
' Declaration
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal onUserCancel As UICancelOption _
)
```

```
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal onUserCancel As UICancelOption _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal networkCredentials As System.Net.ICredentials, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal networkCredentials As System.Net.ICredentials, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal onUserCancel As UICancelOption _
)
```

Paramètres

sourceFileName

String. Chemin d'accès et nom du fichier à transférer. Obligatoire.

address

String ou Uri URL, adresse IP ou URI du serveur de destination. Obligatoire.

userName

String. Nom d'utilisateur pour authentification. La valeur par défaut est une chaîne vide : "".

password

String. Mot de passe pour authentification. La valeur par défaut est une chaîne vide : "".

showUI

Boolean. Si besoin d'afficher l'avancement de l'opération. La valeur par défaut est **False**.

connectionTimeout

Int32. Intervalle de délai en millisecondes. La valeur par défaut est de 100 secondes.

onUserCancel

UICancelOption. Mesures à prendre lorsque l'utilisateur clique sur **Annuler**. La valeur par défaut est ThrowException.

networkCredentials

ICredentials. Informations d'identification pour l'authentification.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le chemin d'accès local n'est pas valide (ArgumentException).
- *connectionTimeout* est inférieur ou égal à zéro (**ArgumentException**).
- *address* ne comporte pas de nom de fichier (InvalidOperationException).
- L'authentification a échoué (SecurityException).
- L'utilisateur n'a pas les autorisations nécessaires (**SecurityException**).
- Le délai de connexion a expiré (TimeoutException).
- Le site Web refuse la demande (WebException).

Notes

Si *showUI* a la valeur **True**, une boîte de dialogue affiche l'avancement de l'opération ; elle comporte un bouton **Annuler** sur lequel l'utilisateur peut cliquer afin d'annuler l'opération. Cette boîte de dialogue n'étant pas modale, elle ne bloque pas les entrées d'utilisateur dans les autres fenêtres du programme.

Si le serveur ne répond pas au cours de l'intervalle spécifié dans *connectionTimeout*, l'opération est annulée et une exception est levée.

My.Computer.Network.UploadFile génère des informations de traçage lorsque vous activez le traçage réseau dans votre application. Pour plus d'informations, consultez Activation du traçage réseau.

Remarque de sécurité
Le protocole FTP envoie les informations, y compris les mots de passe, en texte brut, et ne doit pas être utilisé pour transmettre des informations sensibles.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **My.Computer.Network.UploadFile**.

Pour	**Consultez**
Transférer un fichier	Comment : transférer un fichier dans Visual Basic

Exemple

Cet exemple transfère le fichier `Order.txt` dans `http://www.cohowinery.com/uploads`.

VB

```
My.Computer.Network.UploadFile ( "C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx")
```

Cet exemple transfère le fichier `Order.txt` dans `http://www.cohowinery.com/uploads` sans fournir de nom d'utilisateur ou de mot de passe, affiche la progression du transfert et présente un intervalle de délai d'attente de 500 millisecondes.

VB

```
My.Computer.Network.UploadFile ("C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx","","",True,500)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Network

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Les autorisations suivantes peuvent être nécessaires :

Autorisation	**Description**
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
UIPermission	Contrôle les autorisations relatives aux interfaces utilisateur et au presse-papiers. Énumération associée : AllWindows.
WebPermission	Contrôle les droits d'accès aux ressources Internet HTTP. Énumération associée : **Unrestricted**.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : télécharger un fichier dans Visual Basic

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.Network, objet

System.Uri

System.Net.ICredentials

Microsoft.VisualBasic.FileIO.UICancelOption

Microsoft.VisualBasic.Devices.Network.UploadFile

My.Computer.Ports, objet

Fournit une propriété et une méthode pour accéder aux ports série de l'ordinateur.

Notes

L'objet **My.Computer.Ports** fournit un point d'entrée simple pour accéder à la classe du port série du .NET Framework, SerialPort.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Ports**.

Pour	Consultez
Utiliser un modem connecté à un port série	Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Envoyer une chaîne à un port série	Comment : envoyer des chaînes aux ports série dans Visual Basic
Recevoir des chaînes d'un port série	Comment : recevoir des chaînes des ports série dans Visual Basic
Afficher les ports série disponibles	Comment : afficher les ports série disponibles dans Visual Basic

Exemple

Cet exemple montre comment envoyer des chaînes au port série `COM1` de l'ordinateur.

Le bloc **Using** permet à l'application de fermer le port série même s'il génère une exception. Tout le code qui manipule le port série doit apparaître dans ce bloc ou dans un bloc **Try...Catch...Finally** avec un appel à la méthode Close.

La méthode WriteLine envoie les données au port série.

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

Pour plus d'informations, consultez Comment : envoyer des chaînes aux ports série dans Visual Basic.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Ports

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**

Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Voir aussi

Tâches

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic

Comment : envoyer des chaînes aux ports série dans Visual Basic

Comment : recevoir des chaînes des ports série dans Visual Basic

Comment : afficher les ports série disponibles dans Visual Basic

Référence

My.Computer.Ports, membres de l'objet

My.Computer, objet

Microsoft.VisualBasic.Devices.Ports

My.Computer.Ports, membres de l'objet

L'objet My.Computer.Ports fournit une propriété et une méthode pour accéder aux ports série de l'ordinateur.

Propriétés

SerialPortNames	Obtient une collection des noms des ports série de l'ordinateur. Cette propriété n'est pas disponible pour les applications Web.

Methods

OpenSerialPort	Crée et ouvre un objet SerialPort. Cette méthode n'est pas disponible pour les applications Web.

Voir aussi

Référence

My.Computer.Ports, objet

My.Computer.Ports.OpenSerialPort, méthode

Crée et ouvre un objet SerialPort.

```
' Usage
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te ,parity)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te ,parity ,dataBits)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te ,parity ,dataBits ,stopBits)
' Declaration
Public Function OpenSerialPort( _
   ByVal portName As String _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer, _
   ByVal parity As System.IO.Ports.Parity _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer, _
   ByVal parity As System.IO.Ports.Parity, _
   ByVal dataBits As Integer _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer, _
   ByVal parity As System.IO.Ports.Parity, _
   ByVal dataBits As Integer, _
   ByVal stopBits As System.IO.Ports.StopBits _
) As System.IO.Ports.SerialPort
```

Paramètres

portName

String. Requis. Nom du port à ouvrir.

baudRate

Integer. Vitesse en bauds du port.

parity

Parity. Parité du port.

dataBits

Integer. Paramètre de bit des données du port.

stopBits

StopBits. Paramètre de bit d'arrêt du port.

Valeur de retour

Objet **SerialPort** ouvert, configuré avec les arguments fournis.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- l'argument *portName* a la valeur **Nothing** ou est une chaîne vide (ArgumentNullException) ;
- l'argument *baudRate* ou *dataBits* est nul ou négatif (ArgumentException) ;
- le type d'argument *parity* ne fait pas partie des valeurs de l'énumération **Parity** (InvalidEnumArgumentException) ;
- le type d'argument *stopBits* ne fait pas partie des valeurs de l'énumération **StopBits** (**InvalidEnumArgumentException**).

Notes

La méthode **My.Computer.Ports.OpenSerialPort** crée et ouvre un objet **SerialPort**. Les arguments de la méthode **OpenSerialPort** déterminent les paramètres de l'objet **SerialPort**.

Votre code doit fermer l'objet **SerialPort** lorsqu'il a fini de l'utiliser. Vous pouvez utiliser la méthode Close pour fermer l'objet explicitement ou l'instruction **Using** pour le fermer implicitement. Pour plus d'informations, consultez l'exemple de cette rubrique.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **My.Computer.Ports.OpenSerialPort**.

Pour	**Reportez-vous à**
Utiliser un modem rattaché à un port série	Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Envoyer une chaîne à un port série	Comment : envoyer des chaînes aux ports série dans Visual Basic
Recevoir des chaînes d'un port série	Comment : recevoir des chaînes des ports série dans Visual Basic

Exemple

Cet exemple montre comment envoyer des chaînes au port série `COM1` de l'ordinateur.

Le bloc **Using** permet à l'application de fermer le port série même s'il génère une exception. Tout le code qui manipule le port série doit apparaître dans ce bloc ou dans un bloc **Try...Catch...Finally** avec un appel pour utiliser la méthode **Close**.

La méthode WriteLine envoie les données au port série.

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

Pour plus d'informations, consultez Comment : envoyer des chaînes aux ports série dans Visual Basic.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Ports

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**

Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic
Comment : envoyer des chaînes aux ports série dans Visual Basic
Comment : recevoir des chaînes des ports série dans Visual Basic

Référence

My.Computer.Ports, objet
My.Computer.Ports.SerialPortNames, propriété
Using, instruction (Visual Basic)
System.IO.Ports.Parity
System.IO.Ports.StopBits
System.IO.Ports.SerialPort
Microsoft.VisualBasic.Devices.Ports.OpenSerialPort(System.String)

My.Computer.Ports.SerialPortNames, propriété

Obtient une collection des noms des ports série disponibles sur l'ordinateur.

```
' Usage
Dim value As System.Collections.Generic.ReadOnlyCollection`1(Of String) = My.Computer.Ports
.SerialPortNames
' Declaration
Public ReadOnly Property SerialPortNames As System.Collections.Generic.ReadOnlyCollection`1
(Of String)
```

Valeur de retour

Collection des noms des ports série disponibles sur l'ordinateur.

Notes

La propriété **My.Computer.Ports.SerialPortNames** obtient une collection des noms des ports série disponibles sur l'ordinateur.

Remarque

Les noms de ports retournés par **My.Computer.Ports.SerialPortNames** peuvent être incorrects lorsque vous exécutez Windows 98. Pour éviter des erreurs d'application, utilisez la gestion des exceptions, par exemple l'instruction **Try...Catch...Finally** ou **Using**, lorsque vous utilisez les noms de ports pour ouvrir des ports.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la propriété **My.Computer.Ports.SerialPortNames**.

Pour	Consultez
Afficher les ports série disponibles	Comment : afficher les ports série disponibles dans Visual Basic

Exemple

Cet exemple effectue une boucle sur toutes les chaînes retournées par la propriété **My.Computer.Ports.SerialPortNames**. Ces chaînes représentent les noms des ports série disponibles sur l'ordinateur.

En général, un utilisateur sélectionne le port série que l'application doit utiliser dans la liste des ports disponibles. Dans cet exemple, les noms de ports série sont stockés dans un contrôle ListBox. Pour plus d'informations, consultez ListBox, contrôle (Windows Forms).

VB

```
Sub GetSerialPortNames()
    ' Show all available COM ports.
    For Each sp As String In My.Computer.Ports.SerialPortNames
        ListBox1.Items.Add(sp)
    Next
End Sub
```

Cet exemple nécessite les éléments suivants :

- une référence à l'espace de noms System ;
- un contrôle **ListBox** nommé `ListBox1` dans votre formulaire.

Pour plus d'informations, consultez Comment : afficher les ports série disponibles dans Visual Basic.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Devices

Classe : Ports

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	Non
Service Windows	**Oui**
Site Web	Non

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : afficher les ports série disponibles dans Visual Basic

Référence

My.Computer.Ports, objet
My.Computer.Ports.OpenSerialPort, méthode
Try...Catch...Finally, instruction (Visual Basic)
Using, instruction (Visual Basic)
ReadOnlyCollection
Microsoft.VisualBasic.Devices.Ports.SerialPortNames

My.Computer.Registry, objet

Fournit des propriétés et des méthodes pour manipuler le Registre.

Notes

Pour utiliser ces propriétés, vous devez posséder les autorisations de lecture et d'écriture à partir de RegistryPermissionAccess. Tout code s'exécutant avec une confiance totale (dans la stratégie de sécurité par défaut, tout code installé sur le disque dur local de l'utilisateur) détient les autorisations nécessaires pour accéder au Registre. Pour plus d'informations, consultez RegistryPermission, classe.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	**Consultez**
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Écrire des données à une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Lire des données à partir du Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic

Exemple

Cet exemple lit la valeur `Name` de `HKEY_CURRENT_USER\Software\MyApp` et l'affiche dans un message.

VB

```
Dim readValue As Object
        readValue = My.Computer.Registry.GetValue _
        ("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
        MsgBox("The value is " & CStr(readValue))
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs
Dépannage : manipulation du Registre

Référence

My.Computer.Registry, membres de l'objet
My.Computer.Registry.ClassesRoot, propriété
My.Computer.Registry.CurrentConfig, propriété
My.Computer.Registry.CurrentUser, propriété
My.Computer.Registry.DynData, propriété
My.Computer.Registry.LocalMachine, propriété
My.Computer.Registry.PerformanceData, propriété
My.Computer.Registry.Users, propriété
My.Computer.Registry.GetValue, méthode
My.Computer.Registry.SetValue, méthode
My.Computer, objet

Microsoft.Win32.Registry

Concepts

Tâches courantes relatives au Registre

My.Computer.Registry, membres de l'objet

L'objet **My.Computer.Registry** expose des propriétés et des méthodes permettant d'utiliser les clés de Registre.

Propriétés

ClassesRoot	Correspond à HKEY_CLASSES_ROOT.
CurrentConfig	Correspond à HKEY_CURRENT_CONFIG.
CurrentUser	Correspond à HKEY_CURRENT_USER.
DynData	Correspond à HKEY_DYNAMIC_DATA.
LocalMachine	Correspond à HKEY_LOCAL_MACHINE.
PerformanceData	Correspond à. HKEY_PERFORMANCE_DATA
Users	Correspond à HKEY_USERS.

Methods

GetValue	Obtient une valeur d'une clé de Registre
SetValue	Définit une valeur dans une clé de Registre

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

My.Computer.Registry, objet

Concepts

Tâches courantes relatives au Registre

My.Computer.Registry.ClassesRoot, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_CLASSES_ROOT**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.ClassesRoot
' Declaration
Public ReadOnly Property ClassesRoot As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

L'objet **My.Computer.Registry** expose des méthodes et des propriétés permettant d'utiliser les clés de Registre. Pour plus d'informations, consultez My.Computer.Registry, objet.

HKEY_CLASSES_ROOT est principalement utilisé pour stocker des informations sur des associations de fichier dans Windows.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez les privilèges nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	Description
AllAccess	Accès en création, en lecture et en écriture aux variables du Registre.
Créer	Accès en création aux variables du Registre.
NoAccess	Accès aux variables du Registre non autorisé.
Read	Accès en lecture aux variables du Registre.
Write	Accès en écriture aux variables du Registre.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	Consultez
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple obtient les noms des sous-clés de la clé de Registre `ClassesRoot` et les ajoute à `ListBox1`.

VB

```
Dim keyList As System.Collections.IEnumerable
keyList = My.Computer.Registry.ClassesRoot.GetSubKeyNames()
For Each keyName As String In keyList
   ListBox1.Items.Add(keyName)
Next
```

Cet exemple suppose que vous disposez de **ListBox** nommé `ListBox1` dans votre projet.

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet
Microsoft.Win32.RegistryKey
Microsoft.Win32.Registry.ClassesRoot

Concepts

Tâches courantes relatives au Registre
Sécurité et Registre
Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.CurrentConfig, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_CURRENT_CONFIG**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.CurrentConfig
' Declaration
Public ReadOnly Property CurrentConfig As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

L'objet **My.Computer.Registry** expose des méthodes et des propriétés permettant d'utiliser les clés de Registre. Pour plus d'informations, consultez My.Computer.Registry, objet.

HKEY_CURRENT_CONFIG est utilisé à l'origine pour stocker des paramètres pour les périphériques sur l'ordinateur.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez les privilèges nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	Description
AllAccess	Accès en création, en lecture et en écriture aux variables du Registre.
Créer	Accès en création aux variables du Registre.
NoAccess	Accès aux variables du Registre non autorisé.
Lecture	Accès en lecture aux variables du Registre.
Écriture	Accès en écriture aux variables du Registre.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	Consultez
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple crée la sous-clé `MyDeviceSettings`.

VB

```
My.Computer.Registry.CurrentConfig.CreateSubKey("MyDeviceSettings")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.CurrentConfig

Concepts

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.CurrentUser, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_CURRENT_USER**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.CurrentUser
' Declaration
Public ReadOnly Property CurrentUser As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

L'objet **My.Computer.Registry** expose des méthodes et des propriétés permettant d'utiliser les clés de Registre. Pour plus d'informations, consultez My.Computer.Registry, objet.

HKEY_CURRENT_USER est principalement utilisé pour stocker des paramètres d'utilisateur dans Windows, tels que les couleurs et les polices. La clé diffère en fonction de l'utilisateur.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez le moins de privilèges possibles nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	Description
AllAccess	Accès en création, en lecture et en écriture aux variables du Registre.
Créer	Accès en création aux variables du Registre.
NoAccess	Accès aux variables du Registre non autorisé.
Lecture	Accès en lecture aux variables du Registre.
Écriture	Accès en écriture aux variables du Registre.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	Consultez
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple supprime la sous-clé `Software\MyCompany\Preferences\UserData`.

VB

```
My.Computer.Registry.CurrentUser.DeleteSubKey _
("Software\MyCompany\Preferences\UserData")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet
Microsoft.Win32.RegistryKey
Microsoft.Win32.Registry.CurrentUser

Concepts

Tâches courantes relatives au Registre
Sécurité et Registre
Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.DynData, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_DYNDATA**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.DynData
' Declaration
Public ReadOnly Property DynData As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

L'objet **My.Computer.Registry** expose des méthodes et des propriétés permettant d'utiliser les clés de Registre. Pour plus d'informations, consultez My.Computer.Registry, objet.

HKEY_DYNDATA permet de stocker des données de Registre dynamiques, telles que celles utilisées par les pilotes de périphériques virtuels, et n'est pris en charge que sur les systèmes Windows 95/98. Sur les autres systèmes, l'appel à une méthode de la clé retournée par **Registry.DynData**, telle que **CreateSubKey**, **DeleteSubKey**, **GetSubKeyCount**, **GetValueCount**, etc., lève une exception IOException.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez le moins de privilèges possibles nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	Description
AllAccess	Accès en création, en lecture et en écriture aux variables du Registre.
Créer	Accès en création aux variables du Registre.
NoAccess	Accès aux variables du Registre non autorisé.
Lecture	Accès en lecture aux variables du Registre.
Écriture	Accès en écriture aux variables du Registre.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Vers	Consultez
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminez si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple supprime la sous-clé `Software\MyCompany\Preferences`.

VB

```
My.Computer.Registry.DynData.DeleteSubKeyTree("Software\MyCompany\Preferences")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.DynData

Concepts

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.GetValue, méthode

Obtient une valeur d'une clé de Registre.

```
' Usage
Dim value As Object = My.Computer.Registry.GetValue(keyName ,valueName ,defaultValue)
' Declaration
Public Function GetValue( _
   ByVal keyName As String, _
   ByVal valueName As String, _
   ByVal defaultValue As Object _
) As Object
```

Paramètres

keyName

String. Clé à partir de laquelle la valeur sera récupérée. Requis.

valueName

String. Valeur à récupérer. Requis.

defaultValue

Object. Valeur par défaut à fournir si la valeur n'existe pas. Requis.

Notes

valueName n'est pas sensible à la casse.

Une clé de Registre peut avoir une valeur qui n'est pas associée à un nom. Quand cette valeur sans nom est affichée dans l'Éditeur du Registre, la chaîne "(Default)" apparaît à la place d'un nom. Pour récupérer cette valeur sans nom, spécifiez soit null, soit la chaîne vide ("") pour *valueName*.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le nom de la clé est **Nothing** (ArgumentNullException).
- L'utilisateur n'a pas l'autorisation de lire à partir de clés de Registre (SecurityException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **GetValue**.

Pour	Voir
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic

Exemple

Cet exemple lit la valeur `Name` de `HKEY_CURRENT_USER\Software\MyApp` et l'affiche dans un **MessageBox**.

VB

```
Dim readValue As Object
readValue = My.Computer.Registry.GetValue("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothi
ng)
MsgBox("The value is " & CStr(readValue))
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

GetValue

Concepts

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.LocalMachine, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_LOCAL_MACHINE**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.LocalMachine
' Declaration
Public ReadOnly Property LocalMachine As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

L'objet **My.Computer.Registry** expose des méthodes et des propriétés permettant d'utiliser les clés de Registre. Pour plus d'informations, consultez My.Computer.Registry, objet.

HKEY_LOCAL_MACHINE est habituellement utilisé pour stocker des paramètres Windows qui ne varient pas d'un utilisateur à l'autre. Tous les utilisateurs de l'ordinateur partagent cette clé. Ses cinq sous-clés (**Hardware**, **SAM**, **Security**, **Software**et **System**) stockent les données de configuration pour l'ordinateur local.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez le moins de privilèges possibles nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	Description
AllAccess	Accès en création, en lecture et en écriture aux variables du Registre.
Créer	Accès en création aux variables du Registre.
NoAccess	Accès aux variables du Registre non autorisé.
Lire	Accès en lecture aux variables du Registre.
Écrire	Accès en écriture aux variables du Registre.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	Voir
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple ouvre la sous-clé `Software\MyCompany\Preferences` et définit la valeur `FontColor` sur `"red"`.

VB

```
My.Computer.Registry.LocalMachine.OpenSubKey _
("Software\MyCompany\Preferences", True)
My.Computer.Registry.LocalMachine.SetValue("FontColor", "red")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet
Microsoft.Win32.RegistryKey
Microsoft.Win32.Registry.LocalMachine

Concepts

Tâches courantes relatives au Registre
Sécurité et Registre
Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.PerformanceData, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_PERFORMANCE_DATA**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.PerformanceData
' Declaration
Public ReadOnly Property PerformanceData As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

HKEY_PERFORMANCE_DATA est utilisé pour accéder aux informations sur les performances des composants de logiciel. Lors de son installation, chaque composant logiciel crée des clés pour ses objets ainsi que ses compteurs et écrit des données de compteur pendant son exécution. Ces données ne sont pas stockées dans le Registre, bien que le Registre soit utilisé pour y accéder. À la place, l'appel de fonctions de Registre via **HKEY_PERFORMANCE_DATA** entraîne la collection des données des gestionnaires d'objet système appropriés par le système.

Cette clé n'existe pas sur les systèmes Windows 98 ; si vous tentez de l'utiliser autrement qu'en appelant la méthode **Name**, une exception IOException est levée.

Il est normalement impossible d'écrire dans cette clé ; toute tentative de création d'une sous-clé dessus lève une exception **IOException**.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez le moins de privilèges possibles nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	Description
AllAccess	Accès en création, en lecture et en écriture aux variables du Registre.
Créer	Accès en création aux variables du Registre.
NoAccess	Accès aux variables du Registre non autorisé.
Lecture	Accès en lecture aux variables du Registre.
Écriture	Accès en écriture aux variables du Registre.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	Consultez
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic

Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple obtient la valeur `ThisSoftwar` de la sous-clé `MyCompany`.

VB

```
My.Computer.Registry.PerformanceData.GetValue("MyCompany\ThisSoftware")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.PerformanceData

Concepts

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

My.Computer.Registry.SetValue, méthode

Écrit une valeur dans une clé de Registre.

```
' Usage
My.Computer.Registry.SetValue(keyName ,valueName ,value)
My.Computer.Registry.SetValue(keyName ,valueName ,value ,valueKind)
' Declaration
Public Sub SetValue( _
   ByVal keyName As String, _
   ByVal valueName As String, _
   ByVal value As Object _
)
' -or-
Public Sub SetValue( _
   ByVal keyName As String, _
   ByVal valueName As String, _
   ByVal value As Object, _
   ByVal valueKind As Microsoft.Win32.RegistryValueKind _
)
```

Paramètres

keyName

String. Nom de la clé dans laquelle écrire. Requis.

valueName

String. Nom de la valeur à écrire. Requis.

value

Object. Valeur à écrire. Requis.

valueKind

RegistryValueKind. Requis.

Notes

Si la clé ou la valeur spécifiée n'existe pas, elle est créée.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- Le nom de la clé est **Nothing** (ArgumentNullException).
- Le nom de la clé dépasse la limite de 255 caractères (ArgumentException).
- La ruche spécifiée n'est pas valide (**ArgumentException**).
- La clé est fermée (IOException).
- Le chemin d'accès n'est pas valide (**IOException**).
- La clé de Registre est en lecture seule (UnauthorizedAccessException).

Tâches

Voici un exemple d'une tâche impliquant la méthode **My.Computer.Registry.SetValue**.

Pour	Consultez
Définir une valeur dans une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic

Exemple

Cet exemple affecte le "Nom de l'auteur" à la valeur `Name` dans la clé **HKEY_CURRENT_USER\Software\MyApp**.

VB

```
Dim readValue As Object
readValue = My.Computer.Registry.GetValue _
("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet
Microsoft.Win32.RegistryValueKind
Microsoft.Win32.Registry.SetValue

Concepts

Tâches courantes relatives au Registre
Sécurité et Registre
Lecture et écriture dans le Registre à l'aide de My
Accès au Registre pour les utilisateurs de Visual Basic 6.0

My.Computer.Registry.Users, propriété

Retourne un type RegistryKey qui fournit l'accès à **HKEY_USERS**.

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.Users
' Declaration
Public ReadOnly Property Users As Microsoft.Win32.RegistryKey
```

Valeur de retour

RegistryKey

Notes

L'objet **My.Computer.Registry** expose des méthodes et des propriétés permettant d'utiliser les clés de Registre. Pour plus d'informations, consultez My.Computer.Registry, objet.

HKEY_USERS est utilisé à l'origine pour stocker des paramètres par défaut pour les nouveaux utilisateurs.

La classe RegistryPermission qui se trouve dans l'espace de noms System.Security.Permissions contrôle l'accès aux variables du Registre. Les variables du Registre ne doivent pas être stockées dans des emplacements mémoire auxquels du code sans **RegistryPermission** peut accéder. De même, lorsque vous accordez des autorisations, accordez le moins possible de privilèges nécessaires à l'accomplissement des tâches.

Les valeurs d'accès de l'autorisation Registre sont définies par l'énumération RegistryPermissionAccess. Le tableau suivant décrit en détail ses membres.

Valeur	**Accès aux variables de Registre**
AllAccess	Création, lecture et écriture
Créer	Créer
NoAccess	Aucun accès
Lecture	Lecture
Écriture	Écriture

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Computer.Registry**.

Pour	**Consultez**
Créer une clé de Registre	Comment : créer une clé de Registre et définir ses valeurs dans Visual Basic
Supprimer une clé de Registre	Comment : supprimer une clé de Registre dans Visual Basic
Déterminer si une valeur existe dans une clé de Registre	Comment : déterminer si une valeur existe dans une clé de Registre dans Visual Basic
Lire une valeur d'une clé de Registre	Comment : lire une valeur à partir d'une clé de Registre dans Visual Basic
Définir une valeur dans une clé de Registre	Comment : définir les valeurs dans les clés de Registre dans Visual Basic

Exemple

Cet exemple compte et affiche le nombre de valeurs dans la clé de Registre.

VB

```
Dim keyCount as Integer
keyCount = My.Computer.Registry.Users.ValueCount
MsgBox(keyCount)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.MyServices

Classe : RegistryProxy (fournit l'accès à Registry)

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Dépannage : manipulation du Registre

Référence

My.Computer.Registry, objet

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.Users

Concepts

Tâches courantes relatives au Registre

Sécurité et Registre

Lecture et écriture dans le Registre à l'aide de My

My.Forms, objet

Fournit des propriétés permettant d'accéder à une instance de chaque formulaire Windows déclaré dans le projet actuel.

Notes

L'objet **My.Forms** fournit une instance de chaque formulaire dans le projet actuel. La propriété porte le même que celui du formulaire auquel la propriété accède. Pour plus d'informations sur l'ajout de formulaires à un projet, consultez Comment : ajouter des Windows Forms à un projet.

Vous pouvez accéder aux formulaires fournis par l'objet **My.Forms** à l'aide du nom du formulaire, sans qualification. Étant donné que la propriété porte le même nom que le nom de type du formulaire, vous pouvez accéder à un formulaire comme s'il avait une instance par défaut. Par exemple, `My.Forms.Form1.Show` est équivalent à `Form1.Show`.

L'objet **My.Forms** n'expose que les formulaires associés au projet actuel. Il ne fournit pas un accès aux formulaires déclarés dans les DLL référencées. Pour accéder à un formulaire fourni par une DLL, vous devez utiliser le nom qualifié du formulaire, écrit sous la forme *DllName.FormName*. Pour plus d'informations, consultez Comment : accéder à un formulaire.

La propriété My.Application.OpenForms, propriété permet d'obtenir une collection de tous les formulaires ouverts de l'application.

L'objet et ses propriétés ne sont disponibles que pour les applications Windows.

Propriétés

Chaque propriété de l'objet **My.Forms** fournit l'accès à une instance d'un formulaire du projet actuel. La propriété porte le même nom que celui du formulaire auquel accède la propriété et le type de propriété porte le même nom que le type de formulaire.

Remarque
Si une collision de noms se produit, le nom de la propriété pour accéder à un formulaire est *RootNamespace_Namespace_FormName*. Par exemple, prenons deux formulaires nommés `Form1`. Si l'un de ces formulaires se trouve dans l'espace de noms racine `WindowsApplication1` et dans l'espace de noms `Namespace1`, vous accédez à ce formulaire via `My.Forms.WindowsApplication1_Namespace1_Form1`.

L'objet **My.Forms** fournit un accès à l'instance du formulaire principal de l'application qui a été créé au démarrage. Pour tous les autres formulaires, l'objet **My.Forms** crée une instance du formulaire lorsqu'il est ouvert et le stocke. Les tentatives ultérieures pour accéder à la propriété retournent cette instance du formulaire.

Vous pouvez supprimer un formulaire en assignant **Nothing** à sa propriété. L'accesseur Set de la propriété appelle la méthode Close du formulaire, puis assigne **Nothing** à la valeur stockée. Si vous assignez une valeur autre que **Nothing** à la propriété, l'accesseur Set lève une exception ArgumentException.

Vous pouvez tester si une propriété de l'objet **My.Forms** stocke une instance du formulaire à l'aide de l'opérateur **Is** ou **IsNot**. Vous pouvez utiliser ces opérateurs pour vérifier si la propriété a la valeur **Nothing**.

Remarque
En général, l'opérateur **Is** ou **IsNot** doit lire la valeur de la propriété pour effectuer la comparaison. Toutefois, si la propriété stocke actuellement **Nothing**, la propriété crée une instance du formulaire, puis la retourne. Néanmoins, le compilateur Visual Basic traite différemment les propriétés de l'objet **My.Forms** et permet à l'opérateur **Is** ou **IsNot** de vérifier l'état de la propriété sans modifier sa valeur.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Forms**.

Pour	**Reportez-vous à**
Accéder à un formulaire à partir d'un autre formulaire.	Comment : accéder à un formulaire

Contrôler un formulaire à partir d'un autre formulaire.	Comment : utiliser la communication entre formulaires dans une application

Exemple

Cet exemple modifie le titre du formulaire `SidebarMenu` par défaut.

VB

```
Sub ShowSidebarMenu(ByVal newTitle As String)
    If My.Forms.SidebarMenu IsNot Nothing Then
        My.Forms.SidebarMenu.Text = newTitle
    End If
End Sub
```

Pour que cet exemple fonctionne, votre projet doit avoir un formulaire nommé `SidebarMenu`. Pour plus d'informations, consultez Comment : ajouter des Windows Forms à un projet.

Ce code ne fonctionne que dans un projet d'application Windows.

Configuration requise

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	Non
Application console	Non
Bibliothèque de contrôles Windows	Non
Bibliothèque de contrôles Web	Non
Service Windows	Non
Site Web	Non

Voir aussi

Tâches

Comment : ajouter des Windows Forms à un projet
Comment : accéder à un formulaire

Référence

My.Application.OpenForms, propriété
My.Application.OpenForms, propriété
Is, opérateur (Visual Basic)
IsNot, opérateur
Form
Close

Concepts

Accès aux formulaires de l'application

My.Log, objet

Fournit une propriété et des méthodes pour écrire des informations sur les événements et les exceptions dans les écouteurs de journalisation de l'application.

Notes

L'objet **My.Log** fournit un point d'entrée simple à partir duquel accéder aux services d'enregistrement du .NET Framework. Les méthodes **WriteEntry** et **WriteException** écrivent des messages dans les écouteurs de journalisation de l'application. Les écouteurs peuvent être configurés par le fichier de configuration de l'application. Pour plus d'informations, consultez Procédure pas à pas : modification de l'emplacement des informations My.Application.Log et Utilisation des journaux d'applications dans Visual Basic.

L'objet **My.Log** est uniquement disponible pour les applications ASP.NET. Pour les applications clientes, utilisez **My.Application.Log**. Pour plus d'informations, consultez My.Application.Log, objet.

Configuration requise

Espace de noms : Microsoft.VisualBasic.Logging

Classe : Log

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : écrire des messages de journal
Comment : enregistrer des exceptions dans Visual Basic
Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log

Référence

Membres de l'objet My.Log
My.Application, objet
My.Application.Log, objet
Microsoft.VisualBasic.Logging.Log

Membres de l'objet My.Log

L'objet My.Log fournit une propriété et des méthodes pour écrire des informations sur les événements et les exceptions dans les écouteurs de journalisation de l'application.

Propriétés

Propriété	Description
TraceSource	Accède à l'objet TraceSource sous-jacent de l'objet **My.Log**. Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Méthodes

Méthode	Description
WriteEntry	Écrit un message dans les écouteurs de journalisation de l'application.
WriteException	Écrit des informations sur les exceptions dans les écouteurs de journalisation de l'application.

Voir aussi

Référence

My.Log, objet

My.Request, objet

Obtient l'objet HttpRequest pour la page demandée.

Notes

L'objet **My.Request** contient des informations sur la demande HTTP actuelle.

L'objet **My.Request** est uniquement disponible pour les applications ASP.NET.

Exemple

L'exemple suivant obtient la collection d'en-têtes de l'objet **My.Request** et utilise l'objet **My.Response** pour l'écrire dans la page ASP.NET.

VB

```
<script runat="server">
    Public Sub ShowHeaders()
        ' Load the header collection from the Request object.
        Dim coll As System.Collections.Specialized.NameValueCollection
        coll = My.Request.Headers

        ' Put the names of all keys into a string array.
        For Each key As String In coll.AllKeys
            My.Response.Write("Key: " & key & "<br>")

            ' Get all values under this key.
            For Each value As String In coll.GetValues(key)
                My.Response.Write("Value: " & _
                    Server.HtmlEncode(value) & "<br>")
            Next
        Next
    End Sub
</script>
```

Voir aussi

Référence

My.Response, objet

HttpRequest

My.Response, objet

Obtient l'objet HttpResponse associé à Page. Cet objet vous permet d'envoyer des données de réponse HTTP à un client, et contient des informations sur cette réponse.

Notes

L'objet **My.Response** contient l'objet **HttpResponse** actuel associé à la page.

L'objet **My.Response** est uniquement disponible pour les applications ASP.NET.

Exemple

L'exemple suivant obtient la collection d'en-têtes de l'objet **My.Request** et utilise l'objet **My.Response** pour l'écrire dans la page ASP.NET.

VB

```
<script runat="server">
    Public Sub ShowHeaders()
        ' Load the header collection from the Request object.
        Dim coll As System.Collections.Specialized.NameValueCollection
        coll = My.Request.Headers

        ' Put the names of all keys into a string array.
        For Each key As String In coll.AllKeys
            My.Response.Write("Key: " & key & "<br>")

            ' Get all values under this key.
            For Each value As String In coll.GetValues(key)
                My.Response.Write("Value: " & _
                    Server.HtmlEncode(value) & "<br>")
            Next
        Next
    End Sub
</script>
```

Voir aussi

Référence

My.Request, objet

HttpResponse

My.Resources, objet

Fournit des propriétés et des classes pour accéder aux ressources de l'application.

Notes

L'objet **My.Resources** fournit l'accès aux ressources de l'application et vous permet de récupérer de manière dynamique des ressources pour votre application. Pour plus d'informations, consultez Gestion des ressources de l'application.

L'objet **My.Resources** expose uniquement les ressources globales. Il ne fournit pas un accès aux fichiers de ressources associés aux formulaires. Vous devez accéder aux ressources du formulaire à partir du formulaire. Pour plus d'informations, consultez Procédure pas à pas : localisation de Windows Forms.

Vous pouvez accéder aux fichiers de ressources propres à la culture de l'application à partir de l'objet **My.Resources**. Par défaut, l'objet **My.Resources** recherche des ressources dans le fichier de ressources correspondant à la culture dans la propriété My.Application.UICulture. Toutefois, vous pouvez substituer ce comportement et spécifier une culture particulière à utiliser pour les ressources. Pour plus d'informations, consultez Ressources dans les applications.

Propriétés

Les propriétés de l'objet **My.Resources** fournissent un accès en lecture seule aux ressources de votre application. Pour ajouter ou supprimer des ressources, utilisez le **Concepteur de projets**. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources. Les ressources ajoutées par l'intermédiaire du **Concepteur de projets** sont accessibles à l'aide de **My.Resources.***resourceName*

Vous pouvez également ajouter ou supprimer des fichiers de ressources en sélectionnant votre projet dans l'**Explorateur de solutions** et en cliquant sur **Ajouter un nouvel élément** ou sur **Ajouter un élément existant** dans le menu **Projet**. Les ressources ainsi ajoutées sont disponibles à l'aide de **My.Resources.***resourceFileName.resourceName*.

Chaque ressource a un nom, une catégorie et une valeur, et ces paramètres de ressources déterminent l'affichage de la propriété qui accède à la ressource dans l'objet **My.Resources**. Pour les ressources ajoutées dans le **Concepteur de projets** :

- Le nom détermine le nom de la propriété,
- Les données de ressources représentent la valeur de la propriété,
- La catégorie détermine le type de la propriété :

Catégorie	Type de données de la propriété
Chaînes	String
Images	Bitmap
Icônes	Icon
Audio	UnmanagedMemoryStream La classe **UnmanagedMemoryStream** dérivant de la classe Stream, elle peut être utilisée avec les méthodes qui acceptent les flux, par exemple la méthode My.Computer.Audio.Play.
Fichiers	• String pour les fichiers texte. • **Bitmap** pour les fichiers image. • **Icon** pour les fichiers icône. • **UnmanagedMemoryStream** pour les fichiers audio.

Autres	Déterminé par les informations contenues dans la colonne **Type** du concepteur.

Classes

L'objet **My.Resources** expose chaque fichier de ressources comme une classe contenant des propriétés partagées. Le nom de la classe est identique à celui du fichier de ressources. Comme indiqué dans la section précédente, les ressources contenues dans un fichier de ressources sont exposées comme des propriétés de la classe.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Resources**.

Pour	Consultez
Récupérer une ressource de type chaîne	Comment : récupérer des ressources de type chaîne dans Visual Basic
Récupérer une ressource image	Comment : récupérer des ressources de type image dans Visual Basic
Récupérer une ressource icône	Comment : récupérer des ressources de type icône dans Visual Basic
Récupérer une ressource audio	Comment : récupérer des ressources audio dans Visual Basic
Récupérer une ressource localisée	Comment : récupérer des ressources localisées dans Visual Basic

Exemple

Cet exemple affecte à l'icône du formulaire l'icône nommée `Form1Icon` qui est enregistrée dans le fichier de ressources de l'application.

VB

```
Sub SetFormIcon()
    Me.Icon = My.Resources.Form1Icon
End Sub
```

Pour que cet exemple fonctionne, votre application doit contenir une icône nommée `Form1Icon` dans le fichier de ressources de l'application. Pour plus d'informations, consultez Comment : ajouter ou supprimer des ressources.

Voir aussi

Tâches

Comment : récupérer des ressources de type chaîne dans Visual Basic
Comment : récupérer des ressources de type image dans Visual Basic
Comment : récupérer des ressources de type icône dans Visual Basic
Comment : récupérer des ressources audio dans Visual Basic
Comment : récupérer des ressources localisées dans Visual Basic
Comment : ajouter ou supprimer des ressources
Procédure pas à pas : localisation de Windows Forms

Référence

My.Application.UICulture, propriété

Concepts

Ressources dans les applications

Autres ressources

Gestion des ressources de l'application

My.Settings, objet

Fournit des propriétés et des méthodes pour accéder aux paramètres de l'application.

Notes

L'objet **My.Settings** fournit un accès aux paramètres de l'application et vous permet de stocker et de récupérer de manière dynamique des paramètres de propriété et d'autres informations pour votre application. Pour plus d'informations, consultez Gestion des paramètres de l'application.

Propriétés

Les propriétés de l'objet **My.Settings** fournissent un accès aux paramètres de votre application. Pour ajouter ou supprimer des paramètres, utilisez le **Concepteur de paramètres**. Pour plus d'informations, consultez Comment : ajouter ou supprimer des paramètres d'application.

Chaque paramètre a un **Nom**, un **Type**, une **Portée** et une **Valeur**, et ces paramètres déterminent l'affichage de la propriété qui accède à chaque paramètre dans l'objet **My.Settings**:

- **Nom** détermine le nom de la propriété.
- **Type** détermine le type de la propriété.
- **Portée** indique si la propriété est en lecture seule. Si la valeur est **Application**, la propriété est en lecture seule ; si la valeur est **Utilisateur**, la propriété est en lecture-écriture.
- **Valeur** est la valeur par défaut de la propriété.

Méthodes

Méthode	Description
Reload	Recharge les paramètres utilisateur à partir des dernières valeurs enregistrées.
Save	Enregistre les paramètres utilisateur actuels.

L'objet **My.Settings** fournit également des propriétés et des méthodes avancées héritées de la classe ApplicationSettingsBase.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.Settings**.

Pour	Consultez
Lire un paramètre d'application	Comment : lire des paramètres d'application dans Visual Basic
Modifier un paramètre utilisateur	Comment : modifier les paramètres utilisateur dans Visual Basic
Faire persister les paramètres utilisateur	Comment : rendre persistants les paramètres utilisateur dans Visual Basic
Créer une grille de propriétés pour les paramètres utilisateur	Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic

Exemple

Cet exemple affiche la valeur du paramètre `Nickname`.

VB

```
Sub ShowNickname()
    MsgBox("Nickname is " & My.Settings.Nickname)
End Sub
```

Pour que cet exemple fonctionne, votre application doit contenir un paramètre `Nickname`, de type **String**. Pour plus

d'informations, consultez Comment : ajouter ou supprimer des paramètres d'application.

Voir aussi

Tâches

Comment : lire des paramètres d'application dans Visual Basic

Comment : modifier les paramètres utilisateur dans Visual Basic

Comment : rendre persistants les paramètres utilisateur dans Visual Basic

Comment : créer des grilles de propriétés pour les paramètres utilisateur dans Visual Basic

Comment : ajouter ou supprimer des paramètres d'application

Référence

ApplicationSettingsBase

Autres ressources

Gestion des paramètres de l'application

My.User, objet

Permet d'accéder aux informations concernant l'utilisateur actuel.

Notes

Les propriétés et les méthodes exposées par l'objet **My.User** permettent d'accéder aux informations concernant l'utilisateur actuel. La signification de l'expression « utilisateur actuel » est sensiblement différente selon que vous utilisez une application Windows ou une application Web. Dans une application Windows, l'utilisateur actuel est l'utilisateur qui exécute l'application. Dans une application Web, l'utilisateur actuel est l'utilisateur qui accède à l'application.

La propriété **My.User** permet également d'accéder au IPrincipal pour l'utilisateur actuel. Un objet principal représente le contexte de sécurité de l'utilisateur, y compris l'identité de cet utilisateur et les rôles auxquels il appartient.

Pour les applications Windows, cette propriété fournit les mêmes fonctionnalités que la propriété CurrentPrincipal. Pour les applications Web, cette propriété fournit les mêmes fonctionnalités que la propriété User de l'objet retourné par la propriété Current.

Remarque
Pour les applications Windows, seuls les projets intégrés dans le modèle **Application Windows** initialisent l'objet **My.User** par défaut. Dans tous les autres types de projet Windows, vous devez initialiser l'objet **My.User** en appelant explicitement la méthode My.User.InitializeWithWindowsUser ou en assignant une valeur à **CurrentPrincipal**.

Remarque
L'objet **My.User** ne peut pas rapporter des informations sur l'utilisateur Windows actuel lorsqu'il est exécuté sous Windows 95 et Windows 98 parce que ces systèmes d'exploitation ne prennent pas en charge le concept d'un utilisateur connecté. Pour pouvoir utiliser l'objet **My.User** sur ces systèmes d'exploitation, vous devez implémenter une authentification personnalisée. Pour plus d'informations, consultez Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **My.User**.

Pour	**Reportez-vous à**
Obtenir le nom de connexion de l'utilisateur	Comment : déterminer le nom de connexion d'un utilisateur
Obtenir le nom de domaine de l'utilisateur si l'application utilise l'authentification Windows	Comment : déterminer le domaine de l'utilisateur
Déterminer le rôle de l'utilisateur	Comment : déterminer si un utilisateur est dans un groupe
Implémenter une authentification personnalisée	Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Exemple

Cet exemple vérifie si l'application utilise une authentification personnalisée ou Windows et utilise ces informations pour analyser la propriété **My.User.Name**.

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
```

```
            Dim parts() As String = Split(My.User.Name, "\")
            Dim username As String = parts(1)
            Return username
        Else
            ' The application is using custom authentication.
            Return My.User.Name
        End If
    End Function
```

Configuration requise

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Membres de l'objet My.User

IPrincipal

CurrentPrincipal

User

Current

Membres de l'objet My.User

L'objet My.User fournit un accès aux informations sur l'utilisateur actuel.

Propriétés

Propriété	Description
CurrentPrincipal	Obtient ou définit l'entité de sécurité actuelle (pour une sécurité basée sur les rôles). Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.
IsAuthenticated	Obtient une valeur qui indique si l'utilisateur a été authentifié.
Name	Obtient le nom de l'utilisateur en cours.

Méthodes

Méthode	Description
InitializeWithWindowsUser	Affecte à l'entité de sécurité actuelle du thread la valeur de l'utilisateur Windows qui a démarré l'application. Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.
IsInRole	Détermine si l'utilisateur actuel appartient au rôle spécifié.

Voir aussi

Référence

My.User, objet

My.User.CurrentPrincipal, propriété

Obtient ou définit l'entité de sécurité actuelle (pour la sécurité basée sur les rôles).

```
' Usage
Dim value As System.Security.Principal.IPrincipal = My.User.CurrentPrincipal
' Declaration
Public Property CurrentPrincipal As System.Security.Principal.IPrincipal
```

Valeur de retour

Valeur IPrincipal représentant le contexte de sécurité.

Exceptions

Les conditions ci-dessous peuvent générer une exception :

- l'appelant ne dispose pas des autorisations nécessaires pour définir l'entité de sécurité (SecurityException).

Notes

Vous pouvez affecter à la propriété **My.User.CurrentPrincipal** un objet qui implémente l'interface **IPrincipal** pour activer l'authentification personnalisée.

Dans la plupart des types de projet, cette propriété obtient et définit l'entité de sécurité actuelle du thread. Dans une application ASP.NET, cette propriété obtient et définit les informations de sécurité relatives à l'identité de l'utilisateur de la demande HTTP actuelle.

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Tâches

Pour	**Reportez-vous à**
Obtenir le nom de connexion de l'utilisateur	Comment : déterminer le nom de connexion d'un utilisateur
Obtenir le nom de domaine de l'utilisateur si l'application utilise l'authentification Windows	Comment : déterminer le domaine de l'utilisateur
Implémenter une authentification personnalisée	Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Exemple

Cet exemple vérifie si l'application utilise une authentification personnalisée ou Windows et utilise ces informations pour analyser la propriété **My.User.Name**.

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : User, WebUser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
SecurityPermission	Décrit un ensemble d'autorisations de sécurité appliqué au code. Énumération associée : ControlPrincipal.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer le nom de connexion d'un utilisateur
Comment : déterminer le domaine de l'utilisateur
Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet
System.Security.Principal.IPrincipal
Microsoft.VisualBasic.ApplicationServices.User.CurrentPrincipal

My.User.InitializeWithWindowsUser, méthode

Affecte l'entité de sécurité actuelle du thread à l'utilisateur Windows qui a démarré l'application.

```
' Usage
My.User.InitializeWithWindowsUser()
' Declaration
Public Sub InitializeWithWindowsUser()
```

Notes

Vous pouvez utiliser la méthode **My.User.InitializeWithWindowsUser** pour affecter l'entité de sécurité actuelle du thread à l'utilisateur Windows qui a démarré l'application. Le modèle d'application Visual Basic d'une application Windows appelle par défaut cette méthode au démarrage. Dans d'autres types de projet, vous devez définir l'entité de sécurité actuelle du thread en appelant cette méthode explicitement ou en assignant une valeur à System.Threading.Thread.CurrentPrincipal.

Dans les projets Windows, l'objet **My.User** est basé sur l'entité de sécurité actuelle du thread. Par conséquent, cette méthode peut modifier les informations retournées par **My.User**. Dans une application ASP.NET, l'objet **My.User** est basé sur l'identité de l'utilisateur de la demande HTTP actuelle et n'est pas affecté par cette méthode.

Remarque
Le comportement exact de l'objet **My.User** dépend du type d'application et du système d'exploitation sur lequel s'exécute l'application. Pour plus d'informations, consultez My.User, objet.

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : User, WebUser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
SecurityPermission	Décrit un ensemble d'autorisations de sécurité appliqué au code. Énumération associée : ControlPrincipal.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet

Microsoft.VisualBasic.ApplicationServices.User.InitializeWithWindowsUser

My.User.IsAuthenticated, propriété

Obtient une valeur qui indique si l'utilisateur a été authentifié.

```
' Usage
Dim value As Boolean = My.User.IsAuthenticated
' Declaration
Public ReadOnly Property IsAuthenticated As Boolean
```

Valeur de retour

True si l'utilisateur a été authentifié ; sinon **False**.

Notes

La propriété **My.User.IsAuthenticated** permet à votre code de déterminer si l'utilisateur actuel a été authentifié.

Remarque
Le comportement exact de l'objet **My.User** dépend du type de l'application et du système d'exploitation sur lequel s'exécute l'application. Pour plus d'informations, consultez My.User, objet.

Tâches

Pour	**Consultez**
Implémenter l'authentification personnalisée	Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Exemple

Cet exemple vérifie que l'utilisateur a été authentifié avant d'accéder à une ressource.

VB

```
If My.User.IsAuthenticated Then
    ' Insert code to access a resource here.
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : User, WebUser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	**Disponible**
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**

Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : ControlPrincipal.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet

Microsoft.VisualBasic.ApplicationServices.User.IsAuthenticated

My.User.IsInRole, méthode

Détermine si l'utilisateur actuel appartient au rôle spécifié.

```
' Usage
Dim value As Boolean = My.User.IsInRole(role)
Dim value As Boolean = My.User.IsInRole(role)
' Declaration
Public Function IsInRole( _
   ByVal role As String _
) As Boolean
' -or-
Public Function IsInRole( _
   ByVal role As BuiltInRole _
) As Boolean
```

Paramètres

role

String ou énumération BuiltInRole. Rôle pour lequel vérifier l'appartenance.

Valeur de retour

True si l'utilisateur actuel est un membre du rôle spécifié ; sinon, **False**.

Notes

La méthode **My.User.IsInRole** permet à votre code de déterminer si l'utilisateur actuel est un membre du rôle spécifié.

La surcharge de la méthode **My.User.IsInRole** qui prend une chaîne fournit un accès facile à la méthode IsInRole de l'entité de sécurité actuelle.

La surcharge de la méthode **My.User.IsInRole** qui prend une énumération **BuiltInRole** se comporte différemment, en fonction de l'entité de sécurité actuelle. S'il s'agit de l'entité de sécurité de l'utilisateur Windows (WindowsPrincipal), cette fonction convertit *role* vers l'énumération WindowsBuiltInRole équivalente et retourne le résultat de l'appel de IsInRole. Si l'entité de sécurité est différente, cette fonction passe le nom de la valeur d'énumération de *role* à la méthode **IsInRole** de l'entité de sécurité.

Remarque
Le comportement exact de l'objet **My.User** dépend du type de l'application et du système d'exploitation sur lequel s'exécute l 'application. Pour plus d'informations, consultez My.User, objet.

Tâches

Pour	**Reportez-vous à**
Déterminer le rôle de l'utilisateur	Comment : déterminer si un utilisateur est dans un groupe

Exemple

Cet exemple vérifie si l'utilisateur est un administrateur avant d'accéder à une ressource.

VB

```
If My.User.IsInRole( _
        ApplicationServices.BuiltInRole.Administrator) Then
    ' Insert code to access a resource here.
End If
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : User, WebUser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
SecurityPermission	Décrit un ensemble d'autorisations de sécurité appliqué au code. Énumération associée : ControlPrincipal.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer si un utilisateur est dans un groupe

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet

BuiltInRole, énumération

Microsoft.VisualBasic.ApplicationServices.BuiltInRole

Microsoft.VisualBasic.ApplicationServices.User.IsInRole(System.String)

My.User.Name, propriété

Obtient le nom de l'utilisateur en cours.

```
' Usage
Dim value As String = My.User.Name
' Declaration
Public ReadOnly Property Name As String
```

Valeur de retour

String. Nom de l'utilisateur en cours.

Notes

Vous pouvez utiliser l'objet **My.User** pour obtenir des informations sur l'utilisateur en cours.

L'entité de sécurité qui authentifie l'utilisateur contrôle également le format du nom d'utilisateur. Par défaut, une application utilise l'authentification Windows, et le nom d'utilisateur a le format DOMAIN\USERNAME. Une implémentation personnalisée de l'entité de sécurité n'utilise pas forcément le même format.

Remarque

La propriété **My.User.Name** retourne une chaîne vide lorsque vous utilisez Windows 95 et Windows 98 car ces systèmes d'exploitation ne reconnaissent pas les noms d'utilisateur.

Remarque

Le comportement exact de l'objet **My.User** dépend du type de l'application et du système d'exploitation sur lequel s'exécute l'application. Pour plus d'informations, consultez My.User, objet.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **My.User.Name**.

Pour	Consultez
Obtenir le nom de connexion de l'utilisateur	Comment : déterminer le nom de connexion d'un utilisateur
Obtenir le nom de domaine de l'utilisateur, si l'application utilise l'authentification Windows	Comment : déterminer le domaine de l'utilisateur
Implémenter l'authentification personnalisée	Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Exemple

Cet exemple vérifie si l'application utilise l'authentification Windows ou une authentification personnalisée, et utilise ces informations pour analyser la propriété **My.User.Name**.

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
```

```
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.ApplicationServices

Classe : User, WebUser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	**Oui**

Autorisations

L'autorisation suivante peut être nécessaire :

Autorisation	Description
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : ControlPrincipal.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : déterminer le nom de connexion d'un utilisateur

Comment : déterminer le domaine de l'utilisateur

Procédure pas à pas : implémentation d'une authentification et d'une autorisation personnalisées

Référence

My.User, objet

Microsoft.VisualBasic.ApplicationServices.User.Name

My.WebServices, objet

Fournit des propriétés permettant de créer et d'accéder à une instance unique de chaque service Web XML référencé par le projet en cours.

Notes

L'objet **My.WebServices** fournit une instance de chaque service Web référencé par le projet actuel. Chaque instance est instanciée sur demande. Vous pouvez accéder à ces services Web via les propriétés de l'objet **My.WebServices**. La propriété porte le même nom que celui du service Web auquel la propriété accède. Toute classe qui hérite de SoapHttpClientProtocol est un service Web. Pour plus d'informations sur l'ajout de services Web à un projet, consultez Comment : accéder à un service Web XML en code managé.

L'objet **My.WebServices** n'expose que les services Web associés au projet actuel. Il ne fournit pas un accès aux services Web déclarés dans les DLL référencées. Pour accéder à un service Web fourni par une DLL, vous devez utiliser le nom qualifié du service Web, sous la forme *DllName.WebServiceName*. Pour plus d'informations, consultez Accès aux services Web XML en code managé.

L'objet et ses propriétés ne sont pas disponibles pour les applications Web.

Propriétés

Chaque propriété de l'objet **My.WebServices** fournit l'accès à une instance d'un service Web référencé par le projet actuel. La propriété porte le même que celui du service Web auquel accède la propriété et le type de propriété porte le même nom que le type de service Web.

Remarque

Si une collision de noms se produit, le nom de la propriété pour accéder à un service Web est *RootNamespace_Namespace_FormName*. Par exemple, prenons deux formulaires nommés `Service1`.. Si l'un de ces formulaires se trouve dans l'espace de noms racine `WindowsApplication1` et dans l'espace de noms `Namespace1`, vous accédez à ce service via `My.WebServices.WindowsApplication1_Namespace1_Service1`.

Lorsque vous accédez en premier lieu à l'une des propriétés de l'objet **My.WebServices**, il crée une instance du service Web et le stocke. Les accès ultérieurs de cette propriété retournent cette instance du service Web.

Vous pouvez supprimer un service Web en assignant **Nothing** à la propriété de ce service Web. L'accesseur Set de la propriété assigne **Nothing** à la valeur stockée. Si vous assignez une valeur autre que **Nothing** à la propriété, l'accesseur Set lève une exception ArgumentException.

Vous pouvez tester si une propriété de l'objet **My.WebServices** stocke une instance du service Web à l'aide de l'opérateur **Is** ou **IsNot**. Vous pouvez utiliser ces opérateurs pour vérifier si la propriété a la valeur **Nothing**.

Remarque

En général, l'opérateur **Is** ou **IsNot** doit lire la valeur de la propriété pour effectuer la comparaison. Toutefois, si la propriété stocke actuellement **Nothing**, la propriété crée une instance du service Web, puis la retourne. Néanmoins, le compilateur Visual Basic traite spécialement les propriétés de l'objet **My.WebServices** et permet à l'opérateur **Is** ou **IsNot** de vérifier l'état de la propriété sans modifier sa valeur.

Tâches

Le tableau suivant répertorie un exemple de tâche impliquant l'objet **My.Forms**.

Pour	Reportez-vous à
Appeler un Service Web de façon asynchrone et gérer un événement lorsqu'il s'exécute	Comment : appeler un service Web de manière asynchrone

Exemple

Cet exemple appelle la méthode `FahrenheitToCelsius` du service Web XML `TemperatureConverter` et retourne le résultat.

VB

```
Function ConvertFromFahrenheitToCelsius( _
```

```
        ByVal dFahrenheit As Double) _
        As Double

        Return My.WebServices.TemperatureConverter.FahrenheitToCelsius(dFahrenheit)
    End Function
```

Pour que cet exemple fonctionne, votre projet doit référencer un service Web appelé `Converter` et ce dernier exposer la méthode `ConvertTemperature`. Pour plus d'informations, consultez Comment : accéder à un service Web XML en code managé.

Ce code ne fonctionne pas dans un projet d'application Web.

Configuration requise

Disponibilité par type de projet

Type de projet	Disponible
Application Windows	**Oui**
Bibliothèque de classes	**Oui**
Application console	**Oui**
Bibliothèque de contrôles Windows	**Oui**
Bibliothèque de contrôles Web	**Oui**
Service Windows	**Oui**
Site Web	Non

Voir aussi

Tâches

Comment : accéder à un service Web XML en code managé

Référence

SoapHttpClientProtocol

ArgumentException

Concepts

Accès aux services Web d'une application

Autres ressources

Accès aux services Web XML en code managé

TextFieldParser, objet

Fournit des méthodes et des propriétés pour analyser des fichiers texte structurés.

```
Public Class TextFieldParser
```

Exceptions

La condition ci-dessous peut générer une exception.

- Un champ de texte ne correspond pas au format spécifié. Par exemple, dans un fichier à largeur fixe; l'un des champs ne correspond pas à la largeur spécifiée. (MalformedLineException).

Notes

L'objet **TextFieldParser** fournit des méthodes et des propriétés pour analyser des fichiers texte structurés. Analyser un fichier texte avec le **TextFieldParser** correspond à parcourir un fichier texte, alors que la méthode **ReadFields** pour extraire des champs de texte correspond à fractionner les chaînes.

Le **TextFieldParser** peut analyser deux types de fichiers : les fichiers délimités ou les fichiers à largeur fixe. Certaines propriétés, telles que **Delimiters** et **HasFieldsEnclosedInQuotes** sont significatives uniquement pour les fichiers délimités, alors que la propriété **FieldWidths** est significative uniquement pour les fichiers à largeur fixe.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant l'objet **Microsoft.VisualBasic.FileIO.TextFieldParser**.

Pour	**Voir**
Lire un fichier texte délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier texte à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Lire un fichier texte avec plusieurs formats	Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Exemple

Cet exemple analyse un fichier texte délimité par des tabulations, `Bigfile`.

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser _
("c:\logs\bigfile")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {vbTab}
    Dim currentRow As String()
    'Loop through all of the fields in the file.
    'If any lines are corrupt, report an error and continue parsing.
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadFields()
            ' Include code here to handle the row.
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            " is invalid.  Skipping")
        End Try
    End While
End Using
```

Cet exemple dépend de l'existence d'une fonction, `processFields`, qui traite les champs au cours de la lecture.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Dépannage des exceptions : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

Référence

Membres de l'objet TextFieldParser
TextFieldParser.CommentTokens, propriété
TextFieldParser.Delimiters, propriété
TextFieldParser.EndOfData, propriété
TextFieldParser.ErrorLine, propriété
TextFieldParser.ErrorLineNumber, propriété
TextFieldParser.FieldWidths, propriété
TextFieldParser.HasFieldsEnclosedInQuotes, propriété
TextFieldParser.LineNumber, propriété
TextFieldParser.TextFieldType, propriété
TextFieldParser.TrimWhiteSpace, propriété
TextFieldParser.Close, méthode
TextFieldParser.PeekChars, méthode
TextFieldParser.ReadFields, méthode
TextFieldParser.ReadLine, méthode
TextFieldParser.ReadToEnd, méthode
TextFieldParser.SetDelimiters, méthode
TextFieldParser.SetFieldWidths, méthode
My.Computer.FileSystem.OpenTextFieldParser, méthode
Microsoft.VisualBasic.FileIO.TextFieldParser

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Membres de l'objet TextFieldParser

L'objet **TextFieldParser** fournit des méthodes et des propriétés pour analyser le texte structuré **files**, comme indiqué dans le tableau suivant.

Propriétés

CommentTokens	**String**. Les lignes commençant par le jeton de commentaire spécifié sont ignorées par l'analyseur.
Séparateurs	**String**. Délimiteurs du fichier texte. Valables uniquement pour les fichiers délimités.
EndOfData	**Boolean**. Retourne **True** s'il n'existe aucune ligne non vide et sans commentaire entre la position actuelle du curseur et la fin du fichier.
ErrorLine	**String**. Retourne la ligne qui a provoqué l'exception MalformedLineException la plus récente. La v aleur par défaut est "".
ErrorLineNumber	**Long**. Retourne le numéro de la ligne qui a provoqué l'exception **MalformedLineException** la pl us récente. La valeur par défaut est -1.
FieldWidths	**Integer**. Désigne la largeur de chaque colonne du fichier texte. Valable uniquement pour les fichie rs à largeur fixe.
HasFieldsEnclosedInQuotes	**Boolean**. Retourne **True** si les champs peuvent être mis entre guillemets. Valables uniquement po ur les fichiers délimités. La valeur par défaut est **True**.
LineNumber	**Integer**. Retourne le numéro de ligne actuel, ou *-1* si aucun autre caractère n'est disponible.
TextFieldType	**FieldType**. Type de fichier, délimité ou à largeur fixe. La valeur par défaut est **Delimited**.
TrimWhiteSpace	**Boolean**. Indique si les espaces blancs de début et de fin doivent être retirés des valeurs de cham p.

Méthodes

Close	Ferme le flux sous-jacent.
PeekChars	Lit le nombre de caractères spécifié sans avancer le curseur.
ReadFields	Lit tous les champs de la ligne en cours, les retourne sous la forme d'un tableau de chaînes et déplace le curseu r sur la ligne suivante.
ReadLine	Retourne la ligne en cours sous forme de **String** et déplace le curseur sur la ligne suivante.
ReadToEnd	Lit le reste du flux et le retourne sous forme de **String**.
SetDelimiters	Affecte les valeurs spécifiées aux délimiteurs du lecteur et **Delimited** au type de champ.
SetFieldWidths	Affecte les valeurs spécifiées aux délimiteurs du lecteur et **FixedWidth** au *field type*.

Voir aussi

Référence

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser, constructeur

Initialise une nouvelle instance de la classe **TextFieldParser**.

```
' Usage
Dim value As New TextFieldParser(path)
Dim value As New TextFieldParser(path, defaultEncoding)
Dim value As New TextFieldParser(path, defaultEncoding, detectEncoding)
Dim value As New TextFieldParser(stream)
Dim value As New TextFieldParser(stream, defaultEncoding)
Dim value As New TextFieldParser(stream, defaultEncoding, detectEncoding)
Dim value As New TextFieldParser(stream, defaultEncoding, detectEncoding, leaveOpen)
Dim value As New TextFieldParser(reader)
' Declaration
Public Sub New( _
   ByVal path As String _
)
' -or-
Public Sub New( _
   ByVal path As String, _
   ByVal defaultEncoding As System.Text.Encoding _
)
' -or-
Public Sub New( _
   ByVal path As String, _
   ByVal defaultEncoding As System.Text.Encoding, _
   ByVal detectEncoding As Boolean _
)
' -or-
Public Sub New( _
   ByVal stream As Stream _
)
' -or-
Public Sub New( _
   ByVal stream As Stream, _
   ByVal defaultEncoding As System.Text.Encoding _
)
' -or-
Public Sub New( _
   ByVal stream As Stream, _
   ByVal defaultEncoding As System.Text.Encoding, _
   ByVal detectEncoding As Boolean _
)
' -or-
Public Sub New( _
   ByVal stream As Stream, _
   ByVal defaultEncoding As System.Text.Encoding, _
   ByVal detectEncoding As Boolean, _
   ByVal leaveOpen As Boolean _
)
' -or-
Public Sub New( _
   ByVal reader As TextReader _
)
```

Paramètres

path

String. Nom et chemin d'accès complet du fichier à analyser.

stream

Stream. Flux à analyser.

defaultEncoding

Encoding. Codage de caractères à utiliser si le codage n'est pas déterminé à partir du fichier. La valeur par défaut est UTF8.

detectEncoding

Boolean. Indique s'il convient de rechercher les marques d'ordre d'octet au début du fichier. La valeur par défaut est **True**.

leaveOpen

Boolean. Indique s'il convient de laisser *stream* ouvert lorsque l'objet **TextFieldParser** est fermé. La valeur par défaut est **False**.

reader

TextReader. Flux **TextReader** à analyser.

Notes

Crée un objet **TextFieldParser** pour analyser le fichier ou le flux représenté par les paramètres *path*, *stream* ou *reader*.

Si le paramètre *defaultEncoding* est spécifié, ce constructeur l'utilise comme codage par défaut.

Si le paramètre *detectEncoding* est **True**, ce constructeur essaie de détecter le codage en observant les trois premiers octets du fichier ou flux de données. Il reconnaît automatiquement UTF-8, le texte Unicode little-endian et big-endian si le fichier démarre avec les marques d'ordre d'octet appropriées. Sinon, le codage spécifié par *defaultEncoding* est utilisé.

Attention
Lorsque vous compilez un jeu de caractères avec un paramètre de culture particulier et que vous récupérez ces mêmes carac tères avec un paramètre de culture différent, il se peut que les caractères ne puissent pas être interprétés et provoquent la lev ée d'une exception.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

TextFieldParser, objet
Microsoft.VisualBasic.FileIO.TextFieldParser.#ctor(System.IO.Stream)

TextFieldParser.Close, méthode

Ferme l'objet **TextFieldParser** en cours.

```
' Usage
TextFieldParserObject.Close()
' Declaration
Public Sub Close()
```

Notes

Cette méthode ferme un objet **TextFieldParser**.

Exemple

Cet exemple ferme l'objet **TextFieldParser** `FileReader`.

VB

```
FileReader.Close()
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet

Microsoft.VisualBasic.FileIO.TextFieldParser.Close

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.CommentTokens, propriété

Définit des jetons de commentaire. Un jeton de commentaire est une chaîne qui, lorsqu'elle est placée au début d'une ligne, indique que la ligne est un commentaire et qu'elle doit être ignorée par l'analyseur.

```
' Usage
Dim value As String() = TextFieldParserObject.CommentTokens
' Declaration
Public Property CommentTokens As String()
```

Valeur de retour

String ().

Exceptions

La condition ci-dessous peut générer une exception.

- Un jeton de commentaire contient un espace blanc (ArgumentException).

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Les jetons de commentaire de longueur nulle sont ignorés.

Exemple

Cet exemple spécifie que pour **TextFieldParser**, les lignes `FileReader` commençant par un guillemet simple (') doivent être ignorées.

VB

```
FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
FileReader.CommentTokens = New String() {"'"}
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

TextFieldParser, objet
Microsoft.VisualBasic.FileIO.TextFieldParser.CommentTokens

TextFieldParser.Delimiters, propriété

Définit les délimiteurs d'un fichier texte.

```
' Usage
Dim value As String() = TextFieldParserObject.Delimiters
' Declaration
Public Property Delimiters As String()
```

Valeur de retour

String ().

Exceptions

La condition ci-dessous peut générer une exception :

- Une tentative a été effectuée pour utiliser un caractère de saut de ligne, une chaîne vide ou **Nothing** comme délimiteur (ArgumentException).

Notes

Cette propriété est explicite uniquement si **TextFieldType** = **FieldType.Delimited**.

La méthode **SetDelimiters** permet également de définir les délimiteurs d'un fichier texte. Pour plus d'informations, consultez TextFieldParser.SetDelimiters, méthode.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la propriété **Delimiters**.

Pour	Consultez
Lire un fichier texte délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Exemple

Cet exemple spécifie que le délimiteur de l'objet **TextFieldParser**, `FileReader`, est une virgule (,).

VB

```
FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
FileReader.Delimiters = New String() {","}
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet
TextFieldParser.SetDelimiters, méthode
Microsoft.VisualBasic.FileIO.TextFieldParser.Delimiters

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.EndOfData, propriété

Retourne **True** s'il n'existe aucune ligne non vide et sans commentaire entre la position actuelle du curseur et la fin du fichier.

```
' Usage
Dim value As Boolean = TextFieldParserObject.EndOfData
' Declaration
Public ReadOnly Property EndOfData As Boolean
```

Valeur de retour

Boolean.

Notes

Cette propriété peut être utilisée lors de la lecture de fichiers pour déterminer la fin des données lues.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **EndOfData**.

Pour	Consultez
Lire un fichier délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Exemple

Cet exemple utilise la propriété **EndofData** pour parcourir tous les champs du fichier à l'aide de **TextFieldReader**, `FileReader`.

VB

```
Dim StdFormat As Integer() = {5, 10, 11, -1}
Dim ErrorFormat As Integer() = {5, 5, -1}
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
    FileReader.TextFieldType = FileIO.FieldType.FixedWidth
    FileReader.FieldWidths = StdFormat
    Dim CurrentRow As String()
    While Not FileReader.EndOfData
        Try
            Dim RowType As String = FileReader.PeekChars(3)
            If String.Compare(RowType, "Err") = 0 Then
                ' If this line describes an error, the format of the row will be different.
                FileReader.SetFieldWidths(ErrorFormat)
                CurrentRow = FileReader.ReadFields
                FileReader.SetFieldWidths(StdFormat)
            Else
                ' Otherwise parse the fields normally
                CurrentRow = FileReader.ReadFields
                For Each newString As String In CurrentRow
                    My.Computer.FileSystem.WriteAllText("newFile.txt", newString, True)
                Next
            End If
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Les autorisations suivantes sont requises :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : ControlEvidence.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

TextFieldParser, objet

Microsoft.VisualBasic.FileIO.TextFieldParser.EndOfData

TextFieldParser.ErrorLine, propriété

Retourne la ligne qui a provoqué l'exception MalformedLineException la plus récente.

```
' Usage
Dim value As String = TextFieldParserObject.ErrorLine
' Declaration
Public ReadOnly Property ErrorLine As String
```

Valeur de retour

String.

Notes

Si aucune exception **MalformedLineException** n'a été levée, une chaîne vide est retournée.

La propriété TextFieldParser.ErrorLineNumber peut être utilisée pour afficher le numéro de la ligne qui provoque l'exception.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **ErrorLine**.

Pour	Consultez
Lire un fichier délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Exemple

Cet exemple utilise la propriété **ErrorLine** pour afficher la ligne qui provoque l'exception **MalformedLineException** actuelle.

VB

```
Dim FileReader As Microsoft.VisualBasic.FileIO.TextFieldParser
FileReader = My.Computer.FileSystem.OpenTextFieldParser("C:\test.txt")
Dim currentRow As String()
While Not FileReader.EndOfData
    Try
        currentRow = FileReader.ReadFields
        For Each currentField As String In currentRow
            My.Computer.FileSystem.WriteAllText _
            ("C://testfile.txt", currentField, True)
        Next
    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
        MsgBox("Line " & FileReader.ErrorLine & " is not valid.")
    End Try
End While
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic
Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Référence

TextFieldParser, objet
Microsoft.VisualBasic.FileIO.TextFieldParser.ErrorLine

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.ErrorLineNumber, propriété

Retourne le numéro de la ligne qui a provoqué l'exception MalformedLineException la plus récente.

```
' Usage
Dim value As Long = TextFieldParserObject.ErrorLineNumber
' Declaration
Public ReadOnly Property ErrorLineNumber As Long
```

Valeur de retour

Long.

Exceptions

Cette propriété ne lève pas d'exceptions.

Notes

Si aucune exception **MalformedLineException** n'a été levée, -1 est retourné.

La propriété TextFieldParser.ErrorLine peut être utilisée pour afficher le numéro de la ligne qui provoque l'exception. Les lignes vides et les commentaires ne sont pas ignorés lors de l'identification du numéro de la ligne.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **ErrorLineNumber**.

Pour	Consultez
Lire un fichier délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Exemple

Cet exemple utilise la propriété **ErrorLineNumber** pour afficher l'emplacement de la ligne qui provoque l'exception **MalformedLineException** actuelle.

VB

```
Dim FileReader As Microsoft.VisualBasic.FileIO.TextFieldParser
FileReader = My.Computer.FileSystem.OpenTextFieldParser("C:\test.txt")
Dim currentRow As String()
While Not FileReader.EndOfData
    Try
        currentRow = FileReader.ReadFields
        For Each currentField As String In currentRow
            My.Computer.FileSystem.WriteAllText _
            ("C://testfile.txt", currentField, True)
        Next
    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
        MsgBox("Line " & FileReader.ErrorLineNumber & " is not valid.")
    End Try
End While
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Référence

TextFieldParser, objet

Microsoft.VisualBasic.FileIO.TextFieldParser.ErrorLineNumber

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.FieldWidths, propriété

Désigne la largeur de chaque colonne dans le fichier texte analysé.

```
' Usage
Dim value As Integer() = TextFieldParserObject.FieldWidths
' Declaration
Public Property FieldWidths As Integer()
```

Valeur de retour

Integer ().

Exceptions

La condition ci-dessous peut générer une exception :

- Une valeur de largeur dans un emplacement autre que la dernière entrée du tableau est inférieure ou égale à zéro (ArgumentException).

Notes

Cette propriété est significative uniquement si la propriété TextFieldParser.TextFieldType = **FieldType.FixedWidth**. Si la dernière entrée du tableau est inférieure ou égale à zéro, le champ est supposé être une largeur variable.

La méthode **SetFieldWidths** peut également être utilisée pour définir des largeurs de champ. Pour plus d'informations, consultez TextFieldParser.SetFieldWidths, méthode.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **FieldWidths**.

Pour	Reportez-vous à
Lire un fichier texte à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Exemple

Cet exemple lit le fichier `ParserText.txt` en spécifiant les largeurs. La première colonne a une largeur de 5 caractères, la deuxième 10, la troisième 11, tandis que la quatrième a une largeur variable

VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
    MyReader.FieldWidths = New Integer() {5, 10, 11, -1}
    Dim currentRow As String()
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
                MsgBox(currentField)
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            "is not valid and will be skipped.")
        End Try
    End While
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic
Dépannage des exceptions : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

Référence

TextFieldParser, objet
FieldType, énumération
Microsoft.VisualBasic.FileIO.TextFieldParser.FieldWidths

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.HasFieldsEnclosedInQuotes, propriété

Indique si les champs sont entourés de guillemets lors de l'analyse d'un fichier délimité.

```
' Usage
Dim value As Boolean = TextFieldParserObject.HasFieldsEnclosedInQuotes
' Declaration
Public Property HasFieldsEnclosedInQuotes As Boolean
```

Valeur de retour

Boolean.

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Si la propriété a la valeur **True**, l'analyseur suppose que les champs sont entourés de guillemets (" ") et qu'ils peuvent contenir des fins de ligne.

Si un champ est entouré de guillemets, par exemple `abc, "field2a,field2b", field3` et que cette propriété a la valeur **True**, tout le texte entre guillemets sera retourné tel quel ; cet exemple retournerait `abc|field2a,field2b|field3`. L'affectation de la valeur **False** à cette propriété entraîne le retour de `abc|"field2a|field2b"|field3` par cet exemple.

Exemple

Cet exemple affecte à la propriété **HasFieldsEnclosedInQuotes** de `myReader` la valeur **True**.

VB

```
FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
FileReader.Delimiters = New String() {","}
FileReader.CommentTokens = New String() {""}
FileReader.HasFieldsEnclosedInQuotes = True
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet

Microsoft.VisualBasic.FileIO.TextFieldParser.HasFieldsEnclosedInQuotes

TextFieldParser.LineNumber, propriété

Retourne le numéro de la ligne active, ou retourne -1 si aucun autre caractère n'est disponible dans le flux.

```
' Usage
Dim value As Long = TextFieldParserObject.LineNumber
' Declaration
Public ReadOnly Property LineNumber As Long
```

Valeur de retour

Integer.

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

Les lignes vides et les commentaires ne sont pas ignorés lors de l'identification du numéro de la ligne.

Exemple

Cet exemple recherche le nom « Jones » dans le fichier texte et indique la ligne où il se trouve.

VB

```
Using FileReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    FileReader.Delimiters = New String() {","}
    Dim currentRow As String()
    While Not FileReader.EndOfData
        Try
            currentRow = FileReader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
                If currentField = "Jones" Then
                    MsgBox("The name Jones occurs on line " & _
                    FileReader.LineNumber)
                End If
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
           "is not valid and will be skipped.")
        End Try
    End While
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet

Microsoft.VisualBasic.FileIO.TextFieldParser.LineNumber

TextFieldParser.PeekChars, méthode

Lit le nombre de caractères spécifié sans avancer le curseur.

```
' Usage
Dim value As String = TextFieldParserObject.PeekChars(numberOfChars)
' Declaration
Public Function PeekChars( _
   ByVal numberOfChars As Integer _
) As String
```

Paramètres

numberOfChars

Int32. Nombre de caractères à lire. Obligatoire.

Valeur de retour

String.

Exceptions

La condition ci-dessous peut lever une exception :

- *numberOfChars* est inférieur à 0 (ArgumentException).

Notes

La valeur *numberOfChars* doit être inférieure au nombre total de caractères dans la ligne. Sinon, la chaîne retournée par **PeekChars** est tronquée à la longueur de la ligne.

Les lignes vides sont ignorées.

Les caractères de fin de ligne ne sont pas retournés.

La méthode **PeekChars** n'exécute aucune analyse ; un caractère de fin de ligne contenu dans un champ délimité est interprété comme la fin réelle de la ligne.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **PeekChars**.

Pour	Consultez
Déterminer le format d'un champ avant de l'analyser	Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Exemple

Cet exemple utilise **PeekChars** pour rechercher la fin des données et arrêter l'analyse du fichier à ce stade.

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {","}
    MyReader.CommentTokens = New String() {"'"}
    Dim currentRow As String()
    While (MyReader.PeekChars(1) IsNot "")
        Try
            currentRow = MyReader.ReadFields()
            For Each currentField As String In currentRow
                My.Computer.FileSystem.WriteAllText _
               ("C://testfile.txt", currentField, True)
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
```

```
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Les autorisations suivantes sont requises :

Autorisation	Description
FileIOPermission	Contrôle la possibilité d'accéder à des fichiers et à des dossiers. Énumération associée : Unrestricted.
SecurityPermission	Décrit un jeu d'autorisations de sécurité appliquées au code. Énumération associée : ControlEvidence.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Référence

TextFieldParser, objet

PeekChars

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Autres ressources

Lecture à partir de fichiers en Visual Basic

TextFieldParser.ReadFields, méthode

Lit tous les champs de la ligne en cours, les retourne sous la forme d'un tableau de chaînes et déplace le curseur sur la ligne suivante qui contient des données.

```
' Usage
Dim value As String() = TextFieldParserObject.ReadFields()
' Declaration
Public Function ReadFields() As String()
```

Valeur de retour

String ().

Exceptions

La condition ci-dessous peut lever une exception :

- Un champ ne peut pas être analysé à l'aide du format spécifié (MalformedLineException).

Notes

Pour que les utilisateurs puissent analyser des fichiers texte dans plusieurs formats, la méthode **ReadFields** examine les valeurs de **TextFieldType**, **Delimiters** et **FieldWidths**, si elles sont spécifiées, à chaque appel. Les utilisateurs doivent configurer correctement les propriétés **TextFieldType** et **FieldWidths** ou **Delimiters**, de la manière appropriée. Si **TextFieldType** a la valeur **Delimited** et **Delimiters** n'est pas défini, ou si **TextFieldType** a la valeur **FixedWidth** et **FieldWidths**, une exception est levée.

Si **ReadFields** rencontre des lignes vides, elles sont ignorées et la prochaine ligne non vide est retournée.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la méthode **ReadFields**.

Pour	Consultez
Lire un fichier délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Exemple

Cet exemple utilise la méthode **ReadFields** pour lire le fichier `ParserText.txt` et écrire les champs dans `Testfile.txt`.

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {","}
    Dim currentRow As String()
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadFields()
            For Each currentField As String In currentRow
                My.Computer.FileSystem.WriteAllText _
                ("C://testfile.txt", currentField, True)
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Les autorisations suivantes sont requises :

Autorisation	Description
FileIOPermission	Énumération associée : Unrestricted.
SecurityPermission	Énumération associée : ControlEvidence.

Pour plus d'informations, consultez Sécurité d'accès du code et Demande d'autorisations.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

TextFieldParser, objet
TextFieldParser.TextFieldType, propriété
TextFieldParser.FieldWidths, propriété
TextFieldParser.Delimiters, propriété
TextFieldParser.ReadLine, méthode
Microsoft.VisualBasic.FileIO.TextFieldParser.ReadFields

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.ReadLine, méthode

Retourne la ligne actuelle sous forme de chaîne et avance le curseur sur la ligne suivante.

```
' Usage
Dim value As String = TextFieldParserObject.ReadLine()
' Declaration
Public Function ReadLine() As String
```

Valeur de retour

String

Exceptions

La condition ci-dessous peut lever une exception :

- Le fichier cible n'existe pas (FileNotFoundException).

Notes

La méthode **ReadLine** n'effectue aucune analyse ; un caractère de fin de ligne d'un champ délimité est interprété comme la fin de ligne réelle.

Nothing est retourné si la fin du fichier est atteinte.

Exemple

Cet exemple lit le fichier `ParserText.txt` et l'écrit dans `Testfile.txt`.

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {","}
    Dim currentRow As String
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadLine()
            My.Computer.FileSystem.WriteAllText _
           ("C://testfile.txt", currentRow, True)
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

Si `Testfile.txt` n'existe pas, il est créé par la méthode **WriteAllText**.

Cet exemple écrit les champs sous forme de chaîne unique. Pour que chaque ligne s'affiche dans son propre emplacement dans le fichier de destination, un caractère **VbCrLf** doit être ajouté à la fin de chaque ligne.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet
TextFieldParser.ReadFields, méthode
TextFieldParser.ReadToEnd, méthode
Microsoft.VisualBasic.FileIO.TextFieldParser.ReadLine
Concepts
Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.ReadToEnd, méthode

Lit le reste du fichier texte et le retourne sous forme de chaîne.

```
' Usage
Dim value As String = TextFieldParserObject.ReadToEnd()
' Declaration
Public Function ReadToEnd() As String
```

Valeur de retour

String.

Notes

Il s'agit d'un membre avancé qui ne s'affiche dans IntelliSense que si vous cliquez sur l'onglet **Tous**.

S'il n'y a plus de données à lire car la fin du fichier a été atteinte, **Nothing** est retourné.

La méthode **ReadToEnd** n'ignore pas les lignes vides et les commentaires.

Exemple

Cet exemple utilise la méthode **ReadToEnd** pour lire le fichier entier `ParserText.txt` et l'écrire dans le fichier `Testfile.txt`.

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    Dim allText As String = FileReader.ReadToEnd
    My.Computer.FileSystem.WriteAllText("C://testfile.txt", allText, True)
End Using
```

Si `Testfile.txt` n'existe pas, il est créé par la méthode **WriteAllText**.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet
TextFieldParser.ReadFields, méthode
TextFieldParser.ReadLine, méthode
Microsoft.VisualBasic.FileIO.TextFieldParser.ReadToEnd

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.SetDelimiters, méthode

Affecte les valeurs spécifiées aux délimiteurs du lecteur, et affecte **Delimited** au type de champ.

```
' Usage
TextFieldParserObject.SetDelimiters(delimiters)
' Declaration
Public Sub SetDelimiters( _
   ByVal delimiters As String() _
)
```

Paramètres

delimiters

Tableau de type **String**.

Exceptions

La condition ci-dessous peut lever une exception :

- Un délimiteur est de longueur nulle (ArgumentException).

Notes

Le contenu existant de la propriété **Delimiters** est effacé lors de la définition de cette méthode.

Cette méthode permet de définir des délimiteurs sans créer de tableau.

Tâches

Le tableau suivant présente un exemple de tâche impliquant la méthode **SetDelimiters**.

Pour	Consultez
Analyser un fichier texte	Analyse des fichiers texte avec l'objet TextFieldParser

Exemple

Cet exemple ouvre un analyseur de champ de texte et définit le délimiteur comme `vbTab`.

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\logs\test.log")
    FileReader.SetDelimiters(vbTab)
End Using
```

Remplacez le chemin d'accès `C:\logs\test.log` par le chemin d'accès et le nom du fichier à analyser.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Référence

TextFieldParser, objet

TextFieldParser.SetFieldWidths, méthode

SetDelimiters

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.SetFieldWidths, méthode

Affecte les valeurs spécifiées aux délimiteurs du lecteur et **FixedWidth** au type de champ.

```
' Usage
TextFieldParserObject.SetFieldWidths(fieldWidths)
' Declaration
Public Sub SetFieldWidths( _
   ByVal fieldWidths As Integer() _
)
```

Paramètres
fieldWidths

Tableau de **Integer**.

Exceptions

La condition ci-dessous peut lever une exception :

- Un délimiteur n'est pas un entier (ArgumentException).

Notes

Le contenu existant de la propriété **Delimiters** est effacé lors de la définition de cette méthode.

Cette méthode permet de définir des délimiteurs sans créer de tableau.

Tâches

Le tableau suivant répertorie un exemple de tâches impliquant la méthode **SetFieldWidths**.

Pour	Consultez
Analyser un fichier texte	Analyse des fichiers texte avec l'objet TextFieldParser

Exemple

Cet exemple ouvre un analyseur de champ de texte et affecte la valeur `5` à la largeur de champ.

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\logs\test.log")
    FileReader.SetFieldWidths(5)
End Using
```

Remplacez le chemin d'accès `C:\logs\test.log` par le chemin d'accès et le nom du fichier à analyser.

L'exemple suivant ouvre un analyseur de champ de texte et affecte les valeurs `5`, `10` et variable aux largeurs de champ.

VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\logs\test.log")
    MyReader.SetFieldWidths(5, 10, -1)
End Using
```

Remplacez le chemin d'accès `C:\logs\test.log` par le chemin d'accès et le nom du fichier à analyser.

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Référence

TextFieldParser, objet

TextFieldParser.SetDelimiters, méthode

SetFieldWidths

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.TextFieldType, propriété

Indique si le fichier à analyser est délimité ou a une largeur fixe.

```
' Usage
Dim value As FieldType = TextFieldParserObject.TextFieldType
' Declaration
Public Property TextFieldType As FieldType
```

Valeur de retour

TextFieldType.

Notes

La valeur par défaut de ce paramètre est délimitée.

Tâches

Le tableau suivant répertorie des exemples de tâches impliquant la propriété **TextFieldType**.

Pour	**Reportez-vous à**
Lire un fichier texte délimité	Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Lire un fichier texte à largeur fixe	Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Lire un fichier texte avec plusieurs formats	Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Exemple

Cet exemple crée un **TextFieldParser**, un `FileReader` et spécifie qu'il est délimité.

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = _
    Microsoft.VisualBasic.FileIO.FieldType.Delimited
    FileReader.SetDelimiters(",")
End Using
```

Cet exemple crée un **TextFieldParser**, un `FileReader` et spécifie qu'il a une largeur fixe.

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = _
    Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
    FileReader.FieldWidths = New Integer() {5, 10, 11, -1}
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Dépannage des exceptions : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

Référence

TextFieldParser, objet

FieldType, énumération

FieldType, énumération

TextFieldParser

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser.TrimWhiteSpace, propriété

Indique si l'espace blanc de début et de fin doit être retiré des valeurs de champ.

```
' Usage
Dim value As Boolean = TextFieldParserObject.TrimWhiteSpace
' Declaration
Public Property TrimWhiteSpace As Boolean
```

Valeur de retour

Boolean.

Notes

La valeur par défaut pour cette propriété est **True**.

Exemple

Cet exemple crée un **TextFieldParser**, `FileReader`, et affecte **True** à la propriété **TrimWhiteSpace**.

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = _
    Microsoft.VisualBasic.FileIO.FieldType.Delimited
    FileReader.SetDelimiters(",")
    FileReader.TrimWhiteSpace = True
End Using
```

Configuration requise

Espace de noms : Microsoft.VisualBasic.FileIO

Classe : TextFieldParser

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Autorisations

Aucune autorisation n'est requise.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic
Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

TextFieldParser, objet
My.Computer.FileSystem.OpenTextFieldParser, méthode
Microsoft.VisualBasic.FileIO.TextFieldParser.TrimWhiteSpace

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Opérateurs (Visual Basic)

Dans cette section

Priorité des opérateurs en Visual Basic

Opérateurs répertoriés par fonctionnalité

Types de données des résultats d'opérateur

Opérateurs arithmétiques

Opérateurs d'assignation

Opérateurs de décalage de bits

Opérateurs de comparaison

Opérateurs de concaténation

Opérateurs logiques/de bits

Opérateurs divers

Rubriques connexes

Référence Visual Basic

Visual Basic

Priorité des opérateurs en Visual Basic

Lorsque plusieurs opérations ont lieu dans une même expression, chaque partie est évaluée et résolue dans un ordre prédéterminé. Cet ordre est connu sous le nom de *priorité des opérateurs*.

Règles de priorité

Lorsque les expressions contiennent des opérateurs issus de plusieurs catégories, elles sont évaluées conformément aux règles suivantes :

- Les opérateurs arithmétiques et de concaténation ont l'ordre de priorité décrit ci-dessous et leur priorité est plus élevée que les opérateurs de comparaison, logiques et au niveau du bit.
- Tous les opérateurs de comparaison ont la même priorité, et leur priorité est plus élevée que les opérateurs logiques et de bits, mais moins élevée que les opérateurs arithmétiques et de concaténation.
- Les opérateurs logiques et de bits ont l'ordre de priorité décrit ci-dessous et leur priorité est moins élevée que les opérateurs de comparaison, de concaténation et arithmétiques.
- Les opérateurs ayant la même priorité sont évalués de gauche à droite dans l'ordre où ils apparaissent dans l'expression.

Ordre de priorité

Les opérateurs sont évalués dans l'ordre de priorité suivant :

Opérateurs arithmétiques et de concaténation

Élévation à une puissance (**^**)

Identité et négation unaires (**+**, **–**)

Multiplication et division à virgule flottante (*****, **/**)

Division entière (****)

Modulo arithmétique (**Mod**)

Addition et soustraction (**+**, **–**), concaténation de chaînes (**+**)

Concaténation de chaînes (**&**)

Décalage de bits arithmétique (**<<**, **>>**)

Opérateurs de comparaison

Tous les opérateurs de comparaison (**=**, **<>**, **<**, **<=**, **>**, **>=**, **Is**, **IsNot**, **Like**, **TypeOf**...**Is**)

Opérateurs logiques et au niveau du bit

Négation (**Not**)

Conjonction (**And**, **AndAlso**)

Disjonction inclusive (**Or**, **OrElse**)

Disjonction exclusive (**Xor**)

L'opérateur **=** est uniquement l'opérateur de comparaison d'égalité, pas l'opérateur d'assignation.

L'opérateur de concaténation de chaînes (**&**) n'est pas un opérateur arithmétique, mais il est groupé avec les opérateurs arithmétiques pour établir les priorités.

Les opérateurs **Is** et **IsNot** sont des opérateurs de comparaison de référence d'objet. Ils ne comparent pas les valeurs de deux objets, mais effectuent une vérification pour déterminer si deux variables d'objet désignent la même instance d'objet.

Associativité

Lorsque des opérateurs de même priorité apparaissent dans la même expression, par exemple multiplication et division, le compilateur évalue chaque opération dans l'ordre d'apparition de gauche à droite. L'exemple suivant illustre ce comportement :

```
Dim n1 As Integer = 96 / 8 / 4
```

```
Dim n2 As Integer = (96 / 8) / 4
Dim n3 As Integer = 96 / (8 / 4)
```

La première expression évalue la division *96 / 8* (le résultat est 12), puis la division *12 / 4* (le résultat est 3). Comme le compilateur évalue de gauche à droite les opérations pour `n1`, l'évaluation ne change pas lorsque cet ordre est indiqué explicitement pour `n2`. `n1` et `n2` ont chacun un résultat égal à 3. En revanche, `n3` a un résultat égal à 48, parce que les parenthèses forcent le compilateur à évaluer *8 / 4* en premier.

C'est pourquoi, les opérateurs sont dits *associatif à gauche* en Visual Basic.

Substitution de la priorité et de l'associativité

Vous pouvez utiliser des parenthèses pour forcer certaines parties d'une expression à être évaluée avant d'autres. Elles peuvent servir de substitution à l'ordre de priorité et à l'associativité à gauche. Visual Basic exécute toujours en priorité des opérations situées entre parenthèses. Toutefois, à l'intérieur des parenthèses, il maintient une priorité et une associativité normale, sauf si vous utilisez des parenthèses dans les parenthèses. L'exemple suivant illustre ce comportement :

```
Dim a, b, c, d, e, f, g As Double
a = 8.0
b = 3.0
c = 4.0
d = 2.0
e = 1.0
f = a - b + c / d * e
' The preceding line sets f to 7.0. Because of natural operator
' precedence and associativity, it is exactly equivalent to the
' following line.
f = (a - b) + ((c / d) * e)
' The following line overrides the natural operator precedence
' and left associativity.
g = (a - (b + c)) / (d * e)
' The preceding line sets g to 0.5.
```

Voir aussi

Référence

=, opérateur (Visual Basic)
Is, opérateur (Visual Basic)
IsNot, opérateur
Like, opérateur
TypeOf, opérateur (Visual Basic)
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs et expressions en Visual Basic

Opérateurs répertoriés par fonctionnalité

Consultez l'une des catégories répertoriées ci-dessous ou ouvrez cette partie de la table des matières de l'aide pour consulter une liste alphabétique des opérateurs Visual Basic.

Catégories d'opérateurs

Opérateurs	Description
Opérateurs arithmétiques	Ces opérateurs effectuent des calculs mathématiques.
Opérateurs d'assignation	Ces opérateurs effectuent des opérations d'assignation.
Opérateurs de comparaison	Ces opérateurs effectuent des comparaisons.
Opérateurs de concaténation	Ces opérateurs combinent des chaînes.
Opérateurs logiques/de bits	Ces opérateurs effectuent des opérations logiques.
Opérateurs de décalage de bits	Ces opérateurs effectuent des décalages arithmétiques sur des modèles binaires.
Opérateurs divers	Ces opérateurs effectuent des opérations diverses.

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

Concepts

Opérateurs et expressions en Visual Basic

&, opérateur (Visual Basic)

Génère une concaténation de chaînes de deux expressions.

```
result = expression1 & expression2
```

Éléments

result

Requis. Toute variable **String** ou **Object**.

expression1

Requis. Toute expression avec un type de données qui s'étend à **String**.

expression2

Requis. Toute expression avec un type de données qui s'étend à **String**.

Notes

Si le type de données de *expression1* ou *expression2* n'est pas **String**, mais s'étend à **String**, il est converti en **String**. Si l'un des types de données ne s'étend pas à **String**, le compilateur génère une erreur.

Le type de données de *result* est **String**. Si une ou deux expressions correspondent à Rien ou ont une valeur de System.DBNull.Value, elles sont traitées comme une chaîne avec une valeur de "".

Remarque

L'opérateur **&** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple utilise l'opérateur **&** pour forcer la concaténation de chaînes. Le résultat est une valeur de chaîne représentant la concaténation des deux opérandes de chaîne.

VB

```
Dim sampleStr As String
sampleStr = "Hello" & " World"
' The preceding statement sets sampleStr to "Hello World".
```

Voir aussi

Référence

&=, opérateur (Visual Basic)
Opérateurs de concaténation (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs de concaténation (Visual Basic)

&=, opérateur (Visual Basic)

Concatène une expression **String** à une variable ou une propriété **String** et assigne le résultat à la variable ou à la propriété.

```
variableorproperty &= expression
```

Éléments

variableorproperty

Requis. Toute variable ou propriété **String**.

expression

Requis. Toute expression **String**.

Notes

L'élément situé à gauche de l'opérateur **&=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **&=** assigne la valeur située du côté droit à la variable ou propriété du côté gauche.

Surcharge

L'opérateur &, opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge de l'opérateur **&** affecte le comportement de l'opérateur **&=**. Si votre code utilise **&=** sur une classe ou structure qui surcharge **&**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **&=** pour concaténer deux variables **String** et assigner le résultat à la première variable.

VB

```
Dim var1 As String = "Hello "
Dim var2 As String = "World!"
var1 &= var2
' The value of var1 is now "Hello World!".
```

Voir aussi

Référence

&, opérateur (Visual Basic)
+= (opérateur Visual Basic)
Opérateurs d'assignation
Opérateurs de concaténation (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

*, opérateur (Visual Basic)

Multiplie deux nombres.

```
number1 * number2
```

Éléments
number1

Requis. Toute expression numérique.

number2

Requis. Toute expression numérique.

Résultat

Le résultat est le produit de *number1* et *number2*.

Types pris en charge

Tous les types numériques, y compris les types non signés et à virgule flottante, ainsi que **Decimal**.

Notes

Le type de données du résultat dépend du type des opérandes. Le tableau suivant montre la façon dont le type de données du résultat est déterminé.

Types de données d'opérande	Type de données du résultat
Les deux expressions sont des types de données intégraux (SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong)	Type de données numériques approprié pour les types de données de *number1* et *number2*. Consultez les tableaux « Arithmétique sur les entiers » dans Types de données des résultats d'opérateur.
Les deux expressions sont Decimal	**Decimal**
Les deux expressions sont Single	**Single**
L'une des expressions est un type de données à virgule flottante (**Single** ou Double), mais les deux ne sont pas **Single** (remarquez que **Decimal** n'est pas un type de données à virgule flottante)	**Double**

Si une expression a la valeur Nothing, elle est considérée comme un zéro.

Surcharge

L'opérateur ***** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple utilise l'opérateur ***** pour multiplier deux nombres. Le résultat est le produit des deux opérandes.

VB

```
Dim testValue As Double
testValue = 2 * 2
' The preceding statement sets testValue to 4.
testValue = 459.35 * 334.9
' The preceding statement sets testValue to 153836.315.
```

Voir aussi

Référence

*=, opérateur (Visual Basic)

Opérateurs arithmétiques (Visual Basic)

Priorité des opérateurs en Visual Basic

Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

*=, opérateur (Visual Basic)

Multiplie la valeur d'une variable ou d'une propriété par la valeur d'une expression et assigne le résultat à cette variable ou propriété.

```
variableorproperty *= expression
```

Éléments

variableorproperty

Requis. Toute variable ou propriété numérique.

expression

Requis. Toute expression numérique.

Notes

L'élément situé à gauche de l'opérateur ***=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur ***=** assigne la valeur du côté droit à la variable ou à la propriété du côté gauche.

Surcharge

L'opérateur *, opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge de l'opérateur ***** affecte le comportement de l'opérateur ***=**. Si votre code utilise ***=** sur une classe ou structure qui surcharge *****, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur ***=** pour multiplier une variable **Integer** par une autre et assigner le résultat à la première variable.

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 *= var2
' The value of var1 is now 30.
```

Voir aussi

Référence

*, opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

+, opérateur (Visual Basic)

Ajoute deux nombres ou retourne la valeur positive d'une expression numérique. Peut être également utilisé pour concaténer deux expressions de chaîne.

```
expression1 + expression2
- or -
+ expression1
```

Éléments

expression1

Requis. Toute expression numérique ou de chaîne.

expression2

Requis sauf si l'opérateur **+** calcule une valeur négative. Toute expression numérique ou de chaîne.

Résultat

Si *expression1* et *expression2* sont tous les deux numériques, le résultat est leur somme arithmétique.

Si *expression2* est absent, l'opérateur **+** est l'opérateur d'identité *unaire* pour la valeur inchangée d'une expression. Dans ce sens, l'opération consiste à conserver le signe de *expression1* ; le résultat est donc négatif si *expression1* est négatif.

Si *expression1* et *expression2* sont tous les deux des chaînes, le résultat est la concaténation de leurs valeurs.

Si *expression1* et *expression2* sont de type mixte, l'action effectuée dépend de leur type, de leur contenu et de la définition de l'instruction Option Strict. Pour plus d'informations, consultez les tableaux dans « Remarks ».

Types pris en charge

Tous les types numériques, y compris les types non signés et à virgule flottante, ainsi que **Decimal** et **String**.

Notes

En général, **+** effectue l'addition arithmétique lorsque cela est possible et concatène uniquement lorsque les deux expressions sont des chaînes.

Si aucune expression n'est un **Object**, Visual Basic effectue les actions suivantes.

Types de données d'expressions	**Action du compilateur**
Toutes les expressions sont de types de données numériques (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single** ou **Double**)	Ajoute. Le type de données du résultat est un type numérique approprié aux types de données de *expression1* et *expression2*. Consultez les tableaux « Arithmétique sur les entiers » dans Types de données des résultats d'opérateur.
Toutes les expressions sont de type **String**	Concatène.
Une expression est un type de données numérique et l'autre est une chaîne.	Si **Option Strict** est **On**, une erreur du compilateur est générée. Si **Option Strict** est **Off**, le **String** est implicitement converti en **Double**, puis il est ajouté. Si **String** ne peut pas être converti en **Double**, une exception InvalidCastException est levée.
Une expression est un type de données numérique et l'autre est Nothing (Visual Basic).	Ajoute, avec **Nothing** évaluée à zéro.
Une expression est une chaîne, l'autre est **Nothing**.	Concatène, avec **Nothing** évalué comme "".

Si une expression est une expression **Object**, Visual Basic effectue les actions suivantes.

Types de données d'expressions	Action du compilateur
L'expression **Object** contient une valeur numérique et l'autre est un type de données numérique	Si **Option Strict** est **On**, une erreur du compilateur est générée. Ajoute si **Option Strict** est **Off**.
Object L'expression **Object** contient une valeur numérique et l'autre est de type **String String**	Si **Option Strict** est **On**, une erreur du compilateur est générée. Si **Option Strict** est **Off**, le **String** est implicitement converti en **Double**, puis il est ajouté. Si **String** ne peut pas être converti en **Double**, une exception **InvalidCastException** est levée.
L'expression **Object** contient une chaîne et l'autre est un type de données numérique	Si **Option Strict** est **On**, une erreur du compilateur est générée. Si **Option Strict** est **Off**, la chaîne **Object** est implicitement convertie en **Double**, puis elle est ajoutée. Si la chaîne **Object** ne peut pas être convertie en **Double**, une exception **InvalidCastException** est levée.
Object L'expression **Object** contient une chaîne et l'autre est de type **String String**	Si **Option Strict** est **On**, une erreur du compilateur est générée. Si **Option Strict** est **Off**, **Object** est implicitement converti en **String** et est concaténé.

Si les deux expressions sont des expressions **Object**, Visual Basic effectue les actions suivantes (**Option Strict Off** uniquement).

Types de données d'expressions	Action du compilateur
Toutes les expressions **Object** contiennent des valeurs numériques	Additionne.
Toutes les expressions **Object** sont de type **String**	Concatène.
Une expression **Object** contient une valeur numérique et l'autre contient une chaîne	Convertit implicitement la chaîne **Object** en **Double** et l'ajoute. Si la chaîne **Object** ne peut pas être convertie en une valeur numérique, une exception **InvalidCastException** est levée.

Si l'une des expressions **Object** prend la valeur Nothing ou DBNull, l'opérateur **+** la considère comme un **String** de valeur "".

Remarque
Si vous utilisez l'opérateur **+**, il vous sera parfois difficile de déterminer si une addition ou une concaténation de chaînes se produira. Utilisez l'opérateur **&** pour la concaténation afin d'éliminer toute ambiguïté et de fournir un code auto-documenté.

Surcharge

L'opérateur **+** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple suivant utilise l'opérateur **+** pour ajouter des nombres. Si les opérandes sont tous numériques, Visual Basic calcule le résultat arithmétique. Le résultat arithmétique représente la somme des deux opérandes.

VB

```
Dim sumNumber As Integer
sumNumber = 2 + 2
sumNumber = 4257.04 + 98112
```

```
' The preceding statements set sumNumber to 4 and 102369.
```

Vous pouvez également utiliser l'opérateur **+** pour concaténer des chaînes. Si les opérandes sont tous des chaînes, Visual Basic les concatène. Le résultat de concaténation représente une chaîne unique composée du contenu des deux opérandes l'un après l'autre.

Si les opérandes sont de types mixtes, le résultat dépend de la définition de l'instruction Option Strict. L'exemple suivant illustre le résultat lorsque **Option Strict** est **On**.

VB

```
Option Strict On
```

VB

```
    Dim var1 As String = "34"
    Dim var2 As Integer = 6
    Dim concatenatedNumber As Integer = var1 + var2
...   ' The preceding statement generates a COMPILER ERROR.
```

L'exemple suivant illustre le résultat lorsque **Option Strict** est **Off**.

VB

```
Option Strict Off
```

VB

```
    Dim var1 As String = "34"
    Dim var2 As Integer = 6
    Dim concatenatedNumber As Integer = var1 + var2
...   ' The preceding statement returns 40 after the string in var1 is
    ' converted to a numeric value. This might be an unexpected result.
    ' We do not recommend use of Option Strict Off for these operations.
```

Pour éliminer toute ambiguïté, vous devez utiliser l'opérateur **&** au lieu de **+** pour la concaténation.

Voir aussi

Référence

&, opérateur (Visual Basic)
Opérateurs de concaténation (Visual Basic)
Opérateurs arithmétiques (Visual Basic)
Opérateurs répertoriés par fonctionnalité
Priorité des opérateurs en Visual Basic
Option Strict, instruction

Concepts

Opérateurs arithmétiques en Visual Basic

+= (opérateur Visual Basic)

Ajoute la valeur d'une expression numérique à la valeur d'une variable ou propriété numérique et assigne le résultat à la variable ou propriété. Peut également être utilisé pour concaténer une expression **String** à une variable ou une propriété **String** et assigner le résultat à la variable ou propriété.

```
variableorproperty += expression
```

Éléments

variableorproperty

Requis. Toute variable ou propriété numérique ou **String**.

expression

Requis. Toute expression numérique ou **String**.

Notes

L'élément situé à gauche de l'opérateur **+=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **+=** assigne la valeur du côté droit à la variable ou propriété du côté gauche.

Cet opérateur d'assignation effectue implicitement des conversions étendues, mais pas restrictives, si l'environnement de compilation applique des sémantiques strictes. Pour plus d'informations sur ces conversions, consultez Conversions étendues et restrictives. Pour plus d'informations sur les sémantiques strictes et permissives, consultez Option Strict, instruction.

Si les sémantiques permissives sont autorisées, l'opérateur **+=** exécute implicitement un ensemble de conversions de chaînes et de conversions numériques identiques à celles exécutées par l'opérateur **+**. Pour plus d'informations sur ces conversions, consultez +, opérateur (Visual Basic).

Remarque

Si vous utilisez l'opérateur **+=** il vous sera parfois difficile de déterminer si une addition ou une concaténation de chaînes se produira. Utilisez l'opérateur **&=** pour la concaténation afin d'éliminer toute ambiguïté et de fournir un code auto-document é.

Surcharge

L'opérateur **+** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge de l'opérateur **+** affecte le comportement de l'opérateur **+=**. Si votre code utilise **+=** sur une classe ou structure qui surcharge **+**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **+=** pour combiner la valeur d'une variable à une autre. La première partie utilise **+=** avec des variables numériques pour ajouter une valeur à une autre. La seconde partie utilise **+=** avec des variables **String** pour concaténer une valeur à une autre. Dans les deux cas, le résultat est assigné à la première variable.

VB

```
' This part uses numeric variables.
Dim num1 As Integer = 10
Dim num2 As Integer = 3
num1 += num2
```

VB

```
' This part uses string variables.
Dim str1 As String = "10"
Dim str2 As String = "3"
```

```
str1 += str2
```

La valeur de `num1` est maintenant 13 et la valeur de `str1` est maintenant "103".

Voir aussi

Référence

+, opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs arithmétiques (Visual Basic)
Opérateurs de concaténation (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

=, opérateur (Visual Basic)

Assigne une valeur à une variable ou une propriété.

```
variableorproperty = value
```

Éléments

variableorproperty

Toute variable ou toute propriété accessible en écriture.

value

Tout littéral, constante ou expression.

Notes

L'élément situé à gauche du signe égal (**=**) peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **=** assigne la valeur du côté droit à la variable ou à la propriété du côté gauche.

Remarque

L'opérateur **=** est également utilisé comme un opérateur de comparaison. Pour plus d'informations, consultez Opérateurs de comparaison (Visual Basic).

Surcharge

L'opérateur **=** peut être surchargé uniquement en tant qu'opérateur de comparaison relationnel et non comme un opérateur d'assignation. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant illustre l'opérateur d'assignation. La valeur située à droite est assignée à la variable située à gauche.

VB

```
Dim testInt As Integer
Dim testString As String
Dim testButton As System.Windows.Forms.Button
Dim testObject As Object
testInt = 42
testString = "This is an example of a string literal."
testButton = New System.Windows.Forms.Button()
testObject = testInt
testObject = testString
testObject = testButton
```

Voir aussi

Référence

&=, opérateur (Visual Basic)
*=, opérateur (Visual Basic)
+= (opérateur Visual Basic)
-=, opérateur (Visual Basic)
/=, opérateur (Visual Basic)
\=, opérateur
^=, opérateur (Visual Basic)
Opérateurs de comparaison (Visual Basic)
ReadOnly (Visual Basic)

Concepts

Instructions d'assignation

-, opérateur (Visual Basic)

Retourne la différence entre deux expressions numériques ou la valeur négative d'une expression numérique.

```
expression1 – expression2
- or -
– expression1
```

Éléments

expression1

Requis. Toute expression numérique.

expression2

Requis sauf si l'opérateur **–** calcule une valeur négative. Toute expression numérique.

Résultat

Le résultat constitue la différence entre *expression1* et *expression2* ou la valeur de négation de *expression1*.

Le type de données de résultat est un type numérique approprié aux types de données de *expression1* et *expression2*. Consultez les tableaux « Arithmétique sur les entiers » dans Types de données des résultats d'opérateur.

Types pris en charge

Tous les types numériques, y compris les types non signés et à virgule flottante, ainsi que **Decimal**.

Notes

Dans la première utilisation indiquée dans la syntaxe précédente, l'opérateur **–** est l'opérateur de soustraction arithmétique *binaire* permettant d'obtenir la différence entre deux expressions numériques.

Dans la seconde utilisation indiquée dans la syntaxe précédente, l'opérateur **–** est l'opérateur de négation *unaire* permettant d'obtenir la valeur négative d'une expression. Dans ce sens, la négation consiste à inverser le signe de *expression1* ; le résultat est donc positif si *expression1* est négatif.

Si l'une ou l'autre expression a la valeur Nothing, l'opérateur **–** la considère comme un zéro.

Remarque

L'opérateur **–** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **–** pour calculer et retourner la différence entre deux nombres, puis pour rendre un nombre négatif.

VB

```
Dim binaryResult As Double = 459.35 - 334.9
Dim unaryResult As Double = -334.9
```

Suivant l'exécution de ces instructions, `binaryResult` contient 124,45 et `unaryResult` contient -334,90.

Voir aussi

Référence

-=, opérateur (Visual Basic)
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

-=, opérateur (Visual Basic)

Soustrait la valeur d'une expression de la valeur d'une variable ou d'une propriété et assigne le résultat à la variable ou à la propriété.

```
variableorproperty -= expression
```

Éléments

variableorproperty

Requis. Toute variable ou propriété numérique.

expression

Requis. Toute expression numérique.

Notes

L'élément situé à gauche de l'opérateur **-=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **-=** assigne la valeur du côté droit à la variable ou à la propriété du côté gauche.

Surcharge

-, opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. La surcharge de l'opérateur **-** affecte le comportement de l'opérateur **-=**. Si votre code utilise **-=** sur une classe ou structure qui surcharge **-**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **-=** pour soustraire une variable **Integer** d'une autre et assigner le résultat à la dernière variable.

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 -= var2
' The value of var1 is now 7.
```

Voir aussi

Référence

-, opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

<< opérateur (Visual Basic)

Effectue un décalage arithmétique vers la gauche sur un modèle binaire.

```
result = pattern << amount
```

Éléments

result

Requis. Valeur numérique intégrale. Le résultat du décalage du modèle binaire. Le type de données est le même que celui de *pattern*.

pattern

Requis. Expression numérique intégrale. Modèle binaire qui doit être décalé. Le type de données doit être un type entier (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**).

amount

Requis. Expression numérique. Le nombre de bits pour décaler le modèle binaire. Le type de données doit être **Integer** ou s'étendre à **Integer**.

Notes

Les décalages arithmétiques ne sont pas circulaires, ce qui signifie que les bits décalés à une extrémité du résultat ne sont pas réintroduits à l'autre extrémité. Dans un décalage arithmétique vers la gauche, les bits décalés au-delà de la plage du type de données de résultat sont ignorés, et les positions des bits libérées à droite ont la valeur zéro.

Pour empêcher que le décalage soit effectué par plus de bits que ne peut en contenir le résultat, Visual Basic masque la valeur de *amount* avec un masque de taille correspondant au type de données de *pattern*. L'opérateur binaire AND de ces valeurs est utilisé pour le nombre de positions de décalage. Les masques de taille sont les suivants :

Type de données de *pattern*.	**Masque de taille (décimal)**	**Masque de taille (hexadécimal)**
SByte, **Byte**	7	&H00000007
Short, **UShort**	15	&H0000000F
Integer, **UInteger**	31	&H0000001F
Long, **ULong**	63	&H0000003F

Si la valeur de *amount* est zéro, la valeur de *result* est identique à la valeur de *pattern*. Si la valeur de *amount* est négative, elle est considérée comme une valeur non signée et masquée avec le masque de taille approprié.

Les décalages arithmétiques ne génèrent jamais d'exception de dépassement de capacité.

Remarque

L'opérateur **<<** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **<<** pour effectuer des décalages arithmétiques vers la gauche sur des valeurs intégrales. Le résultat a toujours le même type de données que celui de l'expression décalée.

VB

```
Dim pattern As Short = 192
' The bit pattern is 0000 0000 1100 0000.
Dim result1, result2, result3, result4, result5 As Short
```

```
result1 = pattern << 0
result2 = pattern << 4
result3 = pattern << 9
result4 = pattern << 17
result5 = pattern << -1
```

Les résultats de l'exemple précédent sont les suivants :

- `result1` est 192 (0000 0000 1100 0000).
- `result2` est 3072 (0000 1100 0000 0000).
- `result3` est -32 768 (1000 0000 0000 0000).
- `result4` est 384 (0000 0001 1000 0000).
- `result5` est 0 (décalé de 15 places vers la gauche).

Le nombre de positions de décalage pour `result4` est calculé comme 17 AND 15, ce qui équivaut à 1.

Voir aussi

Référence

Opérateurs de décalage de bits
Opérateurs d'assignation
<<=, opérateur (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

<<=, opérateur (Visual Basic)

Réalise un décalage arithmétique vers la gauche sur la valeur d'une variable ou propriété et assigne le résultat à cette variable ou propriété.

```
variableorproperty <<= amount
```

Éléments

variableorproperty

Requis. Variable ou propriété d'un type intégral (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**).

amount

Requis. Expression numérique d'un type de données qui s'étend à **Integer**.

Notes

L'élément situé à gauche de l'opérateur **<<=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **<<=** assigne la valeur du côté droit à la variable ou à la propriété du côté gauche.

Les décalages arithmétiques ne sont pas circulaires, ce qui signifie que les bits décalés à une extrémité du résultat ne sont pas réintroduits à l'autre extrémité. Dans un décalage arithmétique vers la gauche, les bits décalés au-delà de la plage du type de données de résultat sont ignorés, et les positions de bit libérées à droit sont mises sur zéro.

Surcharge

<< opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. La surcharge de l'opérateur **<<** affecte le comportement de l'opérateur **<<=**. Si votre code utilise **<<=** sur une classe ou structure qui surcharge **<<**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **<<=** pour décaler le modèle binaire d'une variable **Integer** vers la gauche selon les valeurs spécifiées et assigner le résultat à cette variable.

VB

```
Dim var As Integer = 10
Dim shift As Integer = 3
var <<= shift
' The value of var is now 80.
```

Voir aussi

Référence

<< opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs de décalage de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

>>, opérateur (Visual Basic)

Effectue un décalage arithmétique vers la droite sur un modèle binaire.

```
result = pattern >> amount
```

Éléments

result

Requis. Valeur numérique intégrale. Le résultat du décalage du modèle binaire. Le type de données est le même que celui de *pattern*.

pattern

Requis. Expression numérique intégrale. Modèle binaire qui doit être décalé. Le type de données doit être un type entier (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**).

amount

Requis. Expression numérique. Le nombre de bits pour décaler le modèle binaire. Le type de données doit être **Integer** ou s'étendre à **Integer**.

Notes

Les décalages arithmétiques ne sont pas circulaires, ce qui signifie que les bits décalés à une extrémité du résultat ne sont pas réintroduits à l'autre extrémité. Dans un décalage arithmétique vers la droite, les bits décalés au-delà de la position de bit la plus à droite sont supprimés, et le bit (de signe) le plus à gauche est propagé vers les positions de bit libérées à gauche. Ainsi, si la valeur de *pattern* est négative, les positions libérées ont la valeur un ; sinon, elles ont la valeur zéro.

Sachez que les types de données **Byte**, **UShort**, **UInteger** et **ULong** sont considérés comme non signés, et qu'il n'y a donc aucun bit de signe à propager. Si *pattern* est de type non signé, les positions libérées ont toujours la valeur zéro.

Pour empêcher que le décalage soit effectué par plus de bits que ne peut en contenir le résultat, Visual Basic masque la valeur de *amount* avec un masque de taille correspondant au type de données de *pattern*. L'opérateur binaire AND de ces valeurs est utilisé pour le nombre de positions de décalage. Les masques de taille sont les suivants :

Type de données de *pattern*.	**Masque de taille (décimal)**	**Masque de taille (hexadécimal)**
SByte, **Byte**	7	&H00000007
Short, **UShort**	15	&H0000000F
Integer, **UInteger**	31	&H0000001F
Long, **ULong**	63	&H0000003F

Si la valeur de *amount* est zéro, la valeur de *result* est identique à la valeur de *pattern*. Si la valeur de *amount* est négative, elle est considérée comme une valeur non signée et masquée avec le masque de taille approprié.

Les décalages arithmétiques ne génèrent jamais d'exception de dépassement de capacité.

Surcharge

L'opérateur **>>** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **>>** pour effectuer des décalages arithmétiques vers la droite sur des valeurs intégrales. Le résultat a toujours le même type de données que celui de l'expression décalée.

VB

```
Dim pattern As Short = 2560
```

```
' The bit pattern is 0000 1010 0000 0000.
Dim result1, result2, result3, result4, result5 As Short
result1 = pattern >> 0
result2 = pattern >> 4
result3 = pattern >> 10
result4 = pattern >> 18
result5 = pattern >> -1
```

Les résultats de l'exemple précédent sont les suivants :

- `result1` est 2560 (0000 1010 0000 0000).
- `result2` est 160 (0000 0000 1010 0000).
- `result3` est 2 (0000 0000 0000 0010).
- `result4` est 640 (0000 0010 1000 0000).
- `result5` est 0 (décalé de 15 places vers la droite).

Le nombre de positions de décalage pour `result4` est calculé comme 18 AND 15, ce qui équivaut à 2.

L'exemple suivant affiche des décalages arithmétiques sur une valeur négative.

VB

```
Dim negPattern As Short = -8192
' The bit pattern is 1110 0000 0000 0000.
Dim negResult1, negResult2 As Short
negResult1 = negPattern >> 4
negResult2 = negPattern >> 13
```

Les résultats de l'exemple précédent sont les suivants :

- `negresult1` est -512 (1111 1110 0000 0000).
- `negresult2` est -1 (le bit de signe est propagé).

Voir aussi

Référence

Opérateurs de décalage de bits
Opérateurs d'assignation
>>=, opérateur (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

>>=, opérateur (Visual Basic)

Réalise un décalage arithmétique vers la droite sur la valeur d'une variable ou propriété et assigne le résultat à cette variable ou propriété.

```
variableorproperty >>= amount
```

Éléments

variableorproperty

Requis. Variable ou propriété d'un type intégral (**SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**).

amount

Requis. Expression numérique d'un type de données qui s'étend à **Integer**.

Notes

L'élément situé à gauche de l'opérateur **>>=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **>>=** assigne la valeur située du côté droit à la variable ou propriété du côté gauche.

Les décalages arithmétiques ne sont pas circulaires, ce qui signifie que les bits décalés à une extrémité du résultat ne sont pas réintroduits à l'autre extrémité. Dans un décalage arithmétique vers la droite, les bits décalés au-delà de la position de bit la plus à droite sont supprimés, et le bit le plus à gauche est propagé vers les positions de bit libérées à gauche. Ainsi, si *variableorproperty* a une valeur négative, les positions libérées sont mises sur un. Si *variableorproperty* a une valeur positive, ou si son type de données est non signé, les positions libérées sont mises sur zéro.

Surcharge

L'opérateur >>, opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge de l'opérateur **>>** affecte le comportement de l'opérateur **>>=**. Si votre code utilise **>>=** sur une classe ou structure qui surcharge **>>**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **>>=** pour décaler le modèle binaire d'une variable **Integer** vers la droite selon les valeurs spécifiées et assigner le résultat à cette variable.

VB

```
Dim var As Integer = 10
Dim shift As Integer = 2
var >>= shift
' The value of var is now 2 (one bit was lost off the right end).
```

Voir aussi

Référence

>>, opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs de décalage de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

/, opérateur (Visual Basic)

Effectue la division entre deux nombres et retourne un résultat à virgule flottante.

```
expression1 / expression2
```

Éléments

expression1

Requis. Toute expression numérique.

expression2

Requis. Toute expression numérique.

Types pris en charge

Tous les types numériques, y compris les types non signés et à virgule flottante, ainsi que **Decimal**.

Résultat

Le résultat correspond au quotient entier de *expression1* divisé par *expression2*, y compris tout élément restant.

Le \, opérateur retourne le quotient entier, sans le reste.

Notes

Le type de données du résultat dépend du type des opérandes. Le tableau suivant montre la façon dont le type de données du résultat est déterminé.

Types de données d'opérande	**Type de données du résultat**
Les deux expressions sont des types de données intégraux (SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong)	**Double**
Les deux expressions sont du type de données Decimal	**Decimal**
Les deux expressions sont du Single, type de données (Visual Basic)	**Single**
L'une des expressions est un type de données à virgule flottante (Single, type de données (Visual Basic) ou Double, type de données (Visual Basic))	**Double**

Avant d'effectuer la division, toutes les expressions numériques intégrales sont étendues à **Double**. Si vous assignez le résultat à un type de données intégral, Visual Basic essaie de convertir le résultat de **Double** en type de données intégral. Cela peut lever une exception si le résultat ne correspond pas à ce type de données. Consultez en particulier « Essai de division par Zéro » sur cette page de l'aide.

Si *expression1* ou *expression2* correspond à Nothing, elle est considérée comme un zéro.

Essai de division par zéro

Si la valeur de *expression2* est zéro, l'opérateur **/** se comporte différemment pour les divers types de données d'opérande. Le tableau suivant répertorie les comportements possibles.

Types de données d'opérande	**Comportement si la valeur de *expression2* est zéro**
Virgule flottante (**Single** ou **Double**)	Retourne un nombre infini (PositiveInfinity ou NegativeInfinity), ou NaN (pas un nombre) si la valeur de *expression1* est également zéro
Decimal	Lève DivideByZeroException

Intégral (signé ou non signé)	Toute tentative de conversion en type intégral lève une exception OverflowException, car les types intégraux ne peuvent pas accepter **PositiveInfinity**, **NegativeInfinity** ou **NaN**

Remarque

L'opérateur **/** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple utilise l'opérateur **/** pour effectuer une division à virgule flottante. Le résultat est le quotient des deux opérandes.

VB

```
Dim resultValue As Double
resultValue = 10 / 4
resultValue = 10 / 3
```

Les expressions de l'exemple précédent retournent les valeurs 2,5 et 3,333333. Notez que le résultat possède toujours une virgule flottante (**Double**), bien que les deux opérandes soient des constantes entières.

Voir aussi

Référence

/=, opérateur (Visual Basic)
\, opérateur
Types de données des résultats d'opérateur
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

/=, opérateur (Visual Basic)

Divise la valeur d'une variable ou d'une propriété par la valeur d'une expression et assigne le résultat à virgule flottante à cette variable ou propriété.

```
variableorproperty /= expression
```

Éléments

variableorproperty

Requis. Toute variable ou propriété numérique.

expression

Requis. Toute expression numérique.

Notes

L'élément situé à gauche de l'opérateur **/=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **/=** assigne la valeur du côté droit à la variable ou à la propriété du côté gauche.

Cette instruction assigne la valeur **Double** à la variable ou la propriété de gauche. Si **Option Strict** est **On**, *variableorproperty* doit être un **Double**. Si **Option Strict** est **Off**, Visual Basic exécute une conversion implicite et assigne la valeur résultante à *variableorproperty*, avec une erreur possible au moment de l'exécution. Pour plus d'informations, consultez Conversions étendues et restrictives et Option Strict, instruction.

Surcharge

L'opérateur /, opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge de l'opérateur **/** affecte le comportement de l'opérateur **/=**. Si votre code utilise **/=** sur une classe ou structure qui surcharge **/**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **/=** pour diviser une variable **Integer** par une autre et assigner le quotient à la première variable.

VB

```
Dim var1 As Integer = 12
Dim var2 As Integer = 3
var1 /= var2
' The value of var1 is now 4.
```

Voir aussi

Référence

/, opérateur (Visual Basic)
\=, opérateur
Opérateurs d'assignation
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

\ opérateur

Effectue la division de deux nombres et retourne le résultat sous forme d'entier.

```
expression1 \ expression2
```

Éléments
expression1

Requis. Toute expression numérique.

expression2

Requis. Toute expression numérique.

Types pris en charge

Tous les types numériques, y compris les types non signés et à virgule flottante, ainsi que **Decimal**.

Résultat

Le résultat est le quotient entier de *expression1* divisé par *expression2* qui abandonne tout reste et conserve uniquement la partie entière. Ce concept porte le nom de *troncation*.

Le type de données du résultat est un type numérique approprié aux types de données de *expression1* et *expression2*. Consultez les tableaux « Arithmétique sur les entiers » dans Types de données des résultats d'opérateur.

/, opérateur (Visual Basic) retourne le quotient complet qui conserve le reste dans la partie fractionnaire.

Notes

Avant d'exécuter la division, Visual Basic tente de convertir toute expression numérique à virgule flottante en **Long**. Si **Option Strict** est **On**, une erreur du compilateur est générée. Si **Option Strict** est **Off**, un OverflowException est possible si la valeur est à l'extérieur de la plage de Long, type de données (Visual Basic). La conversion en **Long** est également soumise à l'*arrondi bancaire*. Pour plus d'informations, consultez « Parties fractionnaires » dans Fonctions de conversion de types de données.

Si *expression1* ou *expression2* correspond à Nothing, elle est considérée comme un zéro.

Essai de division par zéro

Si *expression2* correspond à zéro, l'opérateur **** lève une exception DivideByZeroException. Cette règle est valable pour tous les types de données numériques des opérandes.

Remarque

L'opérateur **** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple utilise l'opérateur **** pour effectuer une division d'entier. Le résultat est un entier représentant le quotient entier des deux opérandes avec le reste abandonné.

VB

```
Dim resultValue As Integer
resultValue = 11 \ 4
resultValue = 9 \ 3
resultValue = 100 \ 3
resultValue = 67 \ -3
```

Les expressions dans l'exemple précédent retournent respectivement les valeurs 2 ; 3 ; 33 et -22.

Voir aussi
Référence

\=, opérateur
/, opérateur (Visual Basic)
Option Strict, instruction
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

\=, opérateur

Divise la valeur d'une variable ou d'une propriété par la valeur d'une expression et assigne le résultat sous forme d'entier à cette variable ou propriété.

```
variableorproperty \= expression
```

Éléments
variableorproperty

Requis. Toute variable ou propriété numérique.

expression

Requis. Toute expression numérique.

Notes

L'élément situé à gauche de l'opérateur **\=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **\=** assigne la valeur du côté droit à la variable ou propriété du côté gauche.

Pour plus d'informations sur la division par un entier, consultez \, opérateur.

Surcharge

L'opérateur **** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou structure. La surcharge de l'opérateur **** affecte le comportement de l'opérateur **\=**. Si votre code utilise **\=** sur une classe ou structure qui surcharge ****, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **\=** pour diviser une variable **Integer** par une autre et assigner le résultat sous forme d'entier à la première variable.

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 \= var2
' The value of var1 is now 3.
```

Voir aussi
Référence
\, opérateur
/=, opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
Concepts
Instructions d'assignation

^, opérateur (Visual Basic)

Élève un nombre à la puissance d'un autre nombre.

```
number ^ exponent
```

Éléments

number

Requis. Toute expression numérique.

exponent

Requis. Toute expression numérique.

Résultat

Le résultat est *number* élevé à la puissance *exponent*, toujours comme une valeur **Double**.

Types pris en charge

Double. Les opérandes de tous types sont converties en **Double**.

Notes

Visual Basic exécute toujours l'élévation à la puissance dans le Double, type de données (Visual Basic).

La valeur de *exponent* peut être fractionnaire, négative ou les deux.

Lorsque plusieurs élévations à la puissance sont effectuées dans une même expression, l'opérateur **^** est évalué à mesure qu'il est rencontré, de gauche à droite.

Remarque

L'opérateur **^** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **^** pour élever un nombre à la puissance d'un exposant. Le résultat est le premier opérande élevé à la puissance de la deuxième.

VB

```
Dim exp1, exp2, exp3, exp4, exp5, exp6 As Double
exp1 = 2 ^ 2
exp2 = 3 ^ 3 ^ 3
exp3 = (-5) ^ 3
exp4 = (-5) ^ 4
exp5 = 8 ^ (1.0 / 3.0)
exp6 = 8 ^ (-1.0 / 3.0)
```

L'exemple précédent donne les résultats suivants :

`exp1` a la valeur 4 (2 au carré).

`exp2` a la valeur 19 683 (3 au cube, puis cette valeur au cube).

`exp3` a la valeur -125 (-5 au cube).

`exp4` a la valeur 625 (-5 à la puissance quatre).

`exp5` a la valeur 2 (racine cubique de 8).

`exp6` a la valeur 0,5 (1,0 divisé par la racine cubique de 8).

Notez l'importance des parenthèses dans les expressions de l'exemple précédent. À cause de la *priorité des opérateurs*, Visual Basic exécute normalement l'opérateur **^** avant tous les autres, même l'opérateur unaire **–**. Si `exp4` et `exp6` avaient été calculés sans parenthèses, le résultat obtenu serait le suivant :

`exp4 = -5 ^ 4` serait calculé comme -(5 à la puissance quatre), ce qui donnerait -625.

`exp6 = 8 ^ -1.0 / 3.0` serait calculé comme (8 à la puissance -1, ou 0,125) divisé par 3,0, ce qui donnerait 0,041666666666666666666666666666667.

Voir aussi

Référence

^=, opérateur (Visual Basic)
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs arithmétiques en Visual Basic

^=, opérateur (Visual Basic)

Élève la valeur d'une variable ou d'une propriété à la puissance d'une expression et assigne le résultat à cette variable ou propriété.

```
variableorproperty ^= expression
```

Éléments

variableorproperty

Requis. Toute variable ou propriété numérique.

expression

Requis. Toute expression numérique.

Notes

L'élément situé à gauche de l'opérateur **^=** peut être une simple variable scalaire, une propriété ou un élément d'un tableau. La variable ou la propriété ne peut pas être ReadOnly (Visual Basic). L'opérateur **^=** assigne la valeur située du côté droit à la variable ou propriété du côté gauche.

Visual Basic exécute toujours l'élévation à la puissance dans le Double, type de données (Visual Basic). Les opérandes de tous les types différents sont converties en **Double** et le résultat est toujours **Double**.

La valeur de *expression* peut être fractionnaire, négative ou les deux.

Surcharge

L'opérateur ^, opérateur (Visual Basic) peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge de l'opérateur **^** affecte le comportement de l'opérateur **^=**. Si votre code utilise **^=** sur une classe ou structure qui surcharge **^**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **^=** pour élever la valeur d'une variable **Integer** à la puissance d'une seconde variable et assigner le résultat à la première variable.

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 ^= var2
' The value of var1 is now 1000.
```

Voir aussi

Référence

^, opérateur (Visual Basic)
Opérateurs d'assignation
Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

AddressOf, opérateur

Crée une instance déléguée de procédure qui fait référence à la procédure spécifique.

```
AddressOf procedurename
```

Éléments
procedurename

Requis. Spécifie la procédure à référencer par le délégué de procédure créé récemment.

Notes

L'opérateur **AddressOf** crée un délégué de fonction qui pointe vers la fonction spécifiée par *procedurename*. Lorsque la procédure spécifiée est une méthode d'instance, le délégué de fonction fait référence à l'instance et à la méthode. Ensuite, lorsque le délégué de fonction est appelé, la méthode spécifiée de l'instance spécifiée est appelée.

L'opérateur **AddressOf** peut être utilisé comme opérande d'un constructeur délégué ou bien dans un contexte où le type du délégué peut être déterminé par le compilateur.

Exemple

Cet exemple utilise l'opérateur **AddressOf** pour désigner un délégué qui va gérer l'événement **Click** d'un bouton.

VB

```
' Add the following line to Sub Form1_Load().
AddHandler Button1.Click, AddressOf Button1_Click
```

L'exemple suivant utilise l'opérateur **AddressOf** pour désigner la fonction de démarrage d'un thread.

VB

```
Public Sub CountSheep()
    Dim i As Integer = 1 ' Sheep do not count from 0.
    Do While (True) ' Endless loop.
        Console.WriteLine("Sheep " & i & " Baah")
        i = i + 1
        System.Threading.Thread.Sleep(1000) 'Wait 1 second.
    Loop
End Sub

Sub UseThread()
    Dim t As New System.Threading.Thread(AddressOf CountSheep)
    t.Start()
End Sub
```

Voir aussi
Référence
Declare, instruction
Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Concepts
Délégués et opérateur AddressOf

And, opérateur (Visual Basic)

Effectue une conjonction logique sur deux expressions **Boolean** ou une conjonction d'opérations de bits sur deux expressions numériques.

```
result = expression1 And expression2
```

Éléments

result

Requis. Toute expression **Boolean** ou numérique. Pour une comparaison Boolean, *result* est la conjonction logique de deux valeurs **Boolean**. Pour les opérations de bits, *result* est une valeur numérique représentant la conjonction d'opérations de bits de deux modèles binaires numériques.

expression1

Requis. Toute expression **Boolean** ou numérique.

expression2

Requis. Toute expression **Boolean** ou numérique.

Notes

Pour la comparaison Boolean, *result* a la valeur **True** si et seulement si *expression1* et *expression2* ont la valeur **True**. Le tableau suivant illustre la manière dont *result* est déterminé.

Si *expression1* a la valeur	**Et *expression2* a la valeur**	**La valeur de *result* est**
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**False**

Remarque

Dans la comparaison Boolean, l'opérateur **And** évalue toujours les deux expressions qui pourraient inclure des appels de pro cédure. L'opérateur AndAlso effectue un *court-circuit*, ce qui signifie que si *expression1* a la valeur **False**, alors *expression2* n' est pas évalué.

Lorsqu'il est appliqué à des valeurs numériques, l'opérateur **And** effectue une comparaison d'opérations des bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après le tableau suivant :

Si le bit dans *expression1* a la valeur	**Et si le bit dans *expression2* a la valeur**	**Le bit dans *result* a la valeur**
1	1	1
1	0	0
0	1	0
0	0	0

Remarque

Compte tenu que les opérateurs logiques et de bits ont une priorité inférieure à celle des opérateurs arithmétiques et relatio nnels, toutes les opérations au niveau du bit doivent être mises entre parenthèses afin de garantir des résultats précis.

Types de données

Si les opérandes se composent d'une expression **Boolean** et d'une expression numérique, Visual Basic convertit l'expression **Boolean** en une valeur numérique (–1 pour **True** et 0 pour **False**) et exécute une opération au niveau du bit.

Pour une comparaison Boolean, le type de données du résultat est **Boolean**. Pour une comparaison de bits, le type de données du résultat est un type numérique approprié pour les types de données de *expression1* et *expression2*. Consultez le tableau « Relational and Bitwise Comparisons » dans Types de données des résultats d'opérateur.

Remarque

L'opérateur **And** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **And** pour effectuer une conjonction logique sur deux expressions. Le résultat est une valeur **Boolean** qui indique si les deux expressions ont la valeur **True**.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck As Boolean
firstCheck = a > b And b > c
secondCheck = b > a And b > c
```

L'exemple précédent produit respectivement les résultats suivants : **True** et **False**.

L'exemple suivant utilise l'opérateur **And** pour effectuer une conjonction logique sur les bits individuels de deux expressions numériques. Le bit du modèle de résultat est défini si les bits correspondants dans les opérandes ont la valeur 1.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (a And b)
secondPattern = (a And c)
thirdPattern = (b And c)
```

L'exemple précédent produit respectivement les résultats suivants : 8, 2 et 0.

Voir aussi

Référence

Opérateurs logiques/de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
AndAlso, opérateur

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

AndAlso, opérateur

Effectue une conjonction logique "court-circuitante" sur deux expressions.

```
result = expression1 AndAlso expression2
```

Éléments

result

Requis. Toute expression **Boolean**. Le résultat est le résultat **Boolean** de la comparaison des deux expressions.

expression1

Requis. Toute expression **Boolean**.

expression2

Requis. Toute expression **Boolean**.

Notes

Une opération logique est dite *court-circuitante* si le code compilé peut ignorer l'évaluation d'une expression en fonction du résultat d'une autre expression. Si le résultat de la première expression évaluée détermine le résultat final de l'opération, il n'y a pas besoin d'évaluer l'autre expression, car elle ne peut pas changer le résultat final. Un court circuit peut améliorer les performances si l'expression ignorée est complexe ou si elle implique des appels de procédure.

Si les deux expressions ont la valeur **True**, *result* est **True**. Le tableau suivant illustre la manière dont *result* est déterminé.

Si *expression1* a la valeur	**Et *expression2* a la valeur**	**La valeur de *result* est**
True	**True**	**True**
True	**False**	**False**
False	(non évaluée)	**False**

L'une des utilisations de l'opérateur **AndAlso** consiste à tester l'existence d'une instance d'objet avant d'essayer d'accéder à l'un de ses membres. La ligne de code suivante illustre ce comportement.

```
If newObject AndAlso newObject.initFinished Then
```

L'accès à la propriété `initFinished` dans la ligne de code précédente pourrait lever une exception NullReferenceException si la variable `newObject` n'avait pas d'instance d'objet qui lui était assignée. Toutefois, l'opérateur **AndAlso** implique que le compilateur ignore l'accès à `initFinished` si `newObject` a la valeur **Nothing**, parce que **Nothing** correspond à **False**.

Types de données

L'opérateur **AndAlso** est défini uniquement pour Boolean, type de données (Visual Basic). Visual Basic convertit chaque opérande si nécessaire en **Boolean** et exécute l'opération entière en **Boolean**. Si vous assignez le résultat à un type numérique, Visual Basic le convertit de **Boolean** en ce type. Cette conversion pourrait avoir comme conséquence un comportement inattendu. Par exemple, `5 AndAlso 12` donne `-1` lorsqu'il est converti en **Integer**.

Surcharge

Les opérateurs And (Visual Basic) et IsFalse peuvent être *surchargés* ce qui signifie qu'une classe ou une structure peut redéfinir leur comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge des opérateurs **And** et **IsFalse** affecte le comportement de l'opérateur **AndAlso**. Si votre code utilise **AndAlso** sur une classe ou une structure qui surcharge **And** et **IsFalse**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **AndAlso** pour effectuer une conjonction logique sur deux expressions. Le résultat est une valeur **Boolean** qui indique si l'expression conjointe entière est true. Si la première expression est **False**, la seconde n'est pas évaluée.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b AndAlso b > c
secondCheck = b > a AndAlso b > c
thirdCheck = a > b AndAlso c > b
```

L'exemple précédent produit respectivement les résultats suivants : **True**, **False** et **False**. Dans le calcul de `secondCheck`, la seconde expression n'est pas évaluée parce que la première a déjà la valeur **False**. Toutefois, la seconde expression est évaluée dans le calcul de `thirdCheck`.

L'exemple suivant montre une procédure **Function** qui recherche une valeur donnée parmi les éléments d'un tableau. Si le tableau est vide ou si la longueur de tableau a été dépassée, l'instruction **While** ne teste pas l'élément de tableau pour la valeur de recherche.

VB

```
Public Function findValue(ByVal arr() As Double, _
    ByVal searchValue As Double) As Double
    Dim i As Integer = 0
    While i <= UBound(arr) AndAlso arr(i) <> searchValue
        ' If i is greater than UBound(arr), searchValue is not checked.
        i += 1
    End While
    If i > UBound(arr) Then i = -1
    Return i
End Function
```

Voir aussi

Référence

Opérateurs logiques/de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
And, opérateur (Visual Basic)
IsFalse, opérateur

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

GetType, opérateur

Retourne un objet Type pour le type spécifié. L'objet **Type** fournit des informations sur le type telles que ses propriétés, méthodes et événements.

```
GetType(typename)
```

Paramètres
typename

Le nom du type pour lequel vous souhaitez des informations.

Notes

L'opérateur **GetType** retourne l'objet **Type** pour le *typename* spécifié. Vous pouvez transférer le nom de tout type défini dans *typename*. Notamment :

- Type de données Visual Basic, tel que **Boolean** ou **Date**
- Toute classe, structure, module ou interface .NET Framework, telle que System.ArgumentException ou System.Double.
- Toute classe, structure, module ou interface définis par votre application.
- Tout tableau défini par votre application.
- Tout délégué défini par votre application.
- Toute énumération définie par Visual Basic, le .NET Framework ou votre application.

Si vous souhaitez obtenir l'objet de type d'une variable objet, utilisez la méthode System.Type.GetType.

L'opérateur **GetType** peut être utile dans les circonstances suivantes :

- Vous devez accéder aux métadonnées pour un type au moment de l'exécution. L'objet **Type** fournit des métadonnées telles que les membres de type et les informations de déploiement. Vous en avez besoin, par exemple, pour refléter sur un assembly. Pour plus d'informations, consultez System.Reflection.
- Vous souhaitez comparer deux références d'objet pour voir s'ils font référence à des instances du même type. Si tel est le cas, **GetType** retourne des références au même objet **Type**.

Exemple

Voici quelques exemples d'opérateurs **GetType** employés.

VB

```
' The following statement returns the Type object for Integer.
MsgBox(GetType(Integer).ToString())
' The following statement returns the Type object for one-dimensional string arrays.
MsgBox(GetType(String()).ToString())
```

Voir aussi
Référence
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
Concepts
Opérateurs et expressions en Visual Basic

Is, opérateur (Visual Basic)

Compare deux variables référence d'objet.

```
result = object1 Is object2
```

Éléments

result

Requis. Toute valeur **Boolean**.

object1

Requis. Tout nom **Object**.

object2

Requis. Tout nom **Object**.

Notes

L'opérateur **Is** détermine si deux références d'objet font référence au même objet. Toutefois, il n'effectue pas de comparaisons de valeurs. Si *object1* et *object2* font référence à la même instance d'objet, *result* a la valeur **True** ; sinon, *result* a la valeur **False**.

Is peut être utilisé également avec le mot clé **TypeOf** pour créer une expression **TypeOf**...**Is** qui teste si une variable d'objet est compatible avec un type de données.

Exemple

L'exemple suivant utilise l'opérateur **Is** pour comparer des paires de références d'objet. Les résultats sont assignés à une valeur **Boolean** indiquant si les deux objets sont identiques.

VB

```
Dim myObject As New Object
Dim otherObject As New Object
Dim yourObject, thisObject, thatObject As Object
Dim myCheck As Boolean
yourObject = myObject
thisObject = myObject
thatObject = otherObject
' The following statement sets myCheck to True.
myCheck = yourObject Is thisObject
' The following statement sets myCheck to False.
myCheck = thatObject Is thisObject
' The following statement sets myCheck to False.
myCheck = myObject Is thatObject
thatObject = myObject
' The following statement sets myCheck to True.
myCheck = thisObject Is thatObject
```

Comme indiqué dans l'exemple précédent, vous pouvez utiliser l'opérateur **Is** pour tester des objets à liaison anticipée et à liaison tardive.

Voir aussi

Référence

TypeOf, opérateur (Visual Basic)
IsNot, opérateur
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs de comparaison en Visual Basic
Opérateurs et expressions en Visual Basic

IsFalse, opérateur

Détermine si une expression est **False**.

Vous ne pouvez pas appeler explicitement **IsFalse** dans votre code, mais le compilateur Visual Basic peut l'utiliser pour générer le code à partir de clauses **AndAlso**. Si vous définissez une classe ou structure, puis utilisez une variable de ce type dans une clause **AndAlso**, vous devez définir **IsFalse** sur cette classe ou structure.

Le compilateur considère les opérateurs **IsFalse** et **IsTrue** comme une *paire équilibrée*. Cela signifie que si vous définissez l'un, vous devez également définir l'autre.

Remarque

L'opérateur **IsFalse** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsque s on opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez -vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple de code suivant définit le plan d'une structure qui inclut des définitions pour les opérateurs **IsFalse** et **IsTrue**.

VB

```
Public Structure p
    Dim a As Double
    Public Shared Operator IsFalse(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsFalse of w.
        Return b
    End Operator
    Public Shared Operator IsTrue(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsTrue of w.
        Return b
    End Operator
End Structure
```

Voir aussi

Tâches

Comment : définir un opérateur

Référence

IsTrue, opérateur

AndAlso, opérateur

IsNot, opérateur

Compare deux variables référence d'objet.

```
result = object1 IsNot object2
```

Éléments

result

Requis. Valeur **Boolean**.

object1

Requis. Toute variable ou expression **Object**.

object2

Requis. Toute variable ou expression **Object**.

Notes

L'opérateur **IsNot** détermine si deux références d'objet font référence des objets différents. Toutefois, il n'effectue pas de comparaisons de valeurs. Si *object1* et *object2* font référence à la même instance d'objet, *result* a la valeur **False** ; sinon, *result* a la valeur **True**.

IsNot est le contraire de l'opérateur **Is**. **IsNot** a l'avantage de vous faire éviter la syntaxe difficile avec **Not** et **Is**, qui peuvent être difficiles à lire.

Vous pouvez utiliser les opérateurs **Is** et **IsNot** pour tester des objets à liaison anticipée et à liaison tardive.

Exemple

L'exemple de code suivant utilise à la fois l'opérateur **Is** et l'opérateur **IsNot** pour effectuer la même comparaison.

VB

```
Dim o1, o2 As New Object
If Not o1 Is o2 Then MsgBox("o1 and o2 do not refer to the same instance.")
If o1 IsNot o2 Then MsgBox("o1 and o2 do not refer to the same instance.")
```

Voir aussi

Tâches

Comment : déterminer si deux objets sont identiques

Référence

Is, opérateur (Visual Basic)

Priorité des opérateurs en Visual Basic

IsTrue, opérateur

Détermine si une expression est **True**.

Vous ne pouvez pas appeler explicitement **IsTrue** dans votre code, mais le compilateur Visual Basic peut l'utiliser pour générer le code à partir de clauses **OrElse**. Si vous définissez une classe ou structure, puis utilisez une variable de ce type dans une clause **OrElse**, vous devez définir **IsTrue** sur cette classe ou structure.

Le compilateur considère les opérateurs **IsTrue** et **IsFalse** comme une *paire équilibrée*. Cela signifie que si vous définissez l'un, vous devez également définir l'autre.

Utilisation de IsTrue par le compilateur

Lorsque vous avez défini une classe ou structure, vous pouvez utiliser une variable de ce type dans une instruction **For**, **If**, **Else If** ou **While**, ou dans une clause **When**. Si vous faites ceci, le compilateur requiert un opérateur qui convertit votre type en une valeur **Boolean** de façon à pouvoir tester une condition. Il recherche un opérateur adéquat dans l'ordre suivant :

1. Un opérateur de conversion étendue de votre classe ou structure en **Boolean**.
2. L'opérateur **IsTrue** de votre classe ou structure.
3. Un opérateur de conversion restrictive de votre classe ou structure en **Boolean**.

Si vous n'avez pas défini de conversion en **Boolean** ou d'opérateur **IsTrue**, le compilateur signale une erreur.

Remarque

L'opérateur **IsTrue** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsque s on opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez -vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple de code suivant définit le plan d'une structure qui inclut des définitions pour les opérateurs **IsFalse** et **IsTrue**.

VB

```
Public Structure p
    Dim a As Double
    Public Shared Operator IsFalse(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsFalse of w.
        Return b
    End Operator
    Public Shared Operator IsTrue(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsTrue of w.
        Return b
    End Operator
End Structure
```

Voir aussi

Tâches

Comment : définir un opérateur

Référence

IsFalse, opérateur

OrElse, opérateur

Like, opérateur

Compare une chaîne à un modèle.

```
result = string Like pattern
```

Éléments

result

Requis. Toute variable **Boolean**. Le résultat est une valeur **Boolean** indiquant si *string* satisfait *pattern*.

string

Requis. Toute expression **String**.

pattern

Requis. Toute expression **String** se conformant aux critères spéciaux décrits dans les « Notes ».

Notes

Si la valeur de *string* satisfait le modèle contenu dans *pattern*, *result* a la valeur **True**. Si la chaîne ne satisfait pas le modèle, *result* a la valeur **False**. Si *string* et *pattern* sont des chaînes vides, le résultat est **True**.

Méthode de comparaison

Le comportement de l'opérateur **Like** dépend de l'instruction Option Compare. La méthode de comparaison de chaînes par défaut pour chaque fichier source est **Option Compare Binary**.

Options de modèle

La correspondance de modèles intégrée fournit un outil polyvalent pour les comparaisons de chaînes. Les fonctionnalités se conformant aux critères spéciaux vous permettent de faire correspondre chaque caractère d'une *string* à un caractère spécifique, un caractère générique, une liste de caractères ou une plage de caractères. Le tableau suivant illustre les caractères autorisés dans *pattern* et ce à quoi ils correspondent :

Caractères contenus dans *pattern*	**Correspond dans *string* à**
?	Tout caractère unique
*****	Zéro ou plusieurs caractères
#	Tout chiffre (0-9)
[*charlist* **]**	Tout caractère présent dans *charlist*
[! *charlist* **]**	Tout caractère absent de *charlist*

Listes de caractères

Un ou plusieurs caractères (*charlist*) placés entre crochets (**[ ]**) peuvent être utilisés pour faire correspondre tout caractère unique de *string* et peut comprendre presque tous les codes de caractères, y compris les chiffres.

Un point d'exclamation (**!**) au début de *charlist* signifie que la correspondance est trouvée si tous les caractères excepté ceux de *charlist* se trouvent dans *string*. Lorsqu'il est utilisé à l'extérieur de crochets, le point d'exclamation correspond à lui-même.

Caractères spéciaux

Pour faire correspondre les caractères spéciaux comme le crochet ouvrant (**[**), le point d'interrogation (**?**), le signe dièse (**#**) et l'astérisque (*****), vous devez les mettre entre crochets. Le crochet fermant (**]**) ne peut pas être utilisé dans un groupe pour se représenter lui-même, mais il peut être employé comme un caractère spécifique à l'extérieur d'un groupe.

La séquence de caractères `[]` est considérée comme une chaîne de longueur nulle (**""**). Toutefois, elle ne peut pas faire partie d'une liste de caractères entre crochets. Si vous souhaitez vérifier si une position dans la *string* contient un caractère ou pas, vous pouvez utiliser **Like** deux fois. Pour obtenir un exemple, consultez Comment : faire correspondre une chaîne à un modèle.

Plage de caractères

Si vous utilisez un trait d'union (**–**) pour séparer les limites inférieure et supérieure de la plage, *charlist* peut spécifier une plage de caractères. Par exemple, `[A-Z]` trouve une correspondance si la position du caractère correspondante dans *string* contient un caractère dans la plage `A–Z`, et `[!H-L]` trouve une correspondance si la position de caractère correspondante contient un caractère à l'extérieur de la plage `H–L`.

Lorsque vous spécifiez une plage de caractères, ceux-ci doivent apparaître dans l'ordre de tri croissant (c'est-à-dire du premier au dernier). Ainsi, `[A-Z]` est un modèle valide, mais `[Z-A]` n'en est pas un.

Plage de caractères multiples

Pour spécifier plusieurs plages pour la même position, mettez les caractères entre crochets sans délimiteur. Par exemple, `[A-CX-Z]` trouve une correspondance si la position du caractère correspondante dans la *string* contient les caractères compris dans la plage `A–C` ou `X–Z`.

Utilisation du trait d'union

Un trait d'union (**–**) peut apparaître soit au début (après un point d'exclamation, le cas échéant), soit à la fin de *charlist* pour correspondre à lui-même. À tout autre endroit, le trait d'union identifie une plage de caractères délimitée par les caractères de part et d'autre du trait d'union.

Table de classement

La signification d'une plage spécifiée dépend de l'ordre des caractères au moment de l'exécution (comme défini par **Option Compare** et les paramètres régionaux du système sur lequel s'exécute le code). Avec **Option Compare Binary**, la plage `[A-E]` correspond à `A`, `B`, `C`, `D`, et `E`. Avec **Option Compare Text**, `[A-E]` correspond à `A`, `a`, `À`, `à`, `B`, `b`, `C`, `c`, `D`, `d`, `E` et `e`. La plage ne correspond pas `Ê` ou `ê` parce que les caractères accentués sont classés après les caractères inaccentués dans l'ordre de tri.

Caractères digrammes

Dans certaines langues, certains caractères alphabétiques représentent deux caractères séparés. Par exemple, plusieurs langues utilisent le caractère `æ` pour représenter le caractère `a` et le caractère `e` lorsqu'ils apparaissent ensemble. L'opérateur **Like** reconnaît l'équivalence du caractère digramme simple et des deux caractères individuels.

Lorsqu'une langue qui utilise un caractère digramme est spécifiée dans les paramètres régionaux du système, une occurrence du caractère digramme simple dans *pattern* ou *string* correspond à la séquence équivalente de deux caractères dans l'autre chaîne. De même, un caractère digramme dans *pattern* placé entre crochets (tout seul, dans une liste ou dans une plage) correspond à la séquence équivalente de deux caractères dans *string*.

Surcharge

L'opérateur **Like** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple utilise l'opérateur **Like** pour comparer des chaînes de plusieurs modèles. Les résultats se dirigent vers une variable **Boolean** qui indique si chaque chaîne satisfait le modèle.

VB

```
Dim testCheck As Boolean
' The following statement returns True (does "F" satisfy "F"?)
testCheck = "F" Like "F"
' The following statement returns False for Option Compare Binary
'    and True for Option Compare Text (does "F" satisfy "f"?)
testCheck = "F" Like "f"
' The following statement returns False (does "F" satisfy "FFF"?)
testCheck = "F" Like "FFF"
' The following statement returns True (does "aBBBa" have an "a" at the
'    beginning, an "a" at the end, and any number of characters in
'    between?)
testCheck = "aBBBa" Like "a*a"
' The following statement returns True (does "F" occur in the set of
'    characters from "A" through "Z"?)
testCheck = "F" Like "[A-Z]"
' The following statement returns False (does "F" NOT occur in the
'    set of characters from "A" through "Z"?)
testCheck = "F" Like "[!A-Z]"
```

```
' The following statement returns True (does "a2a" begin and end with
'    an "a" and have any single-digit number in between?)
testCheck = "a2a" Like "a#a"
' The following statement returns True (does "aM5b" begin with an "a",
'    followed by any character from the set "L" through "P", followed
'    by any single-digit number, and end with any character NOT in
'    the character set "c" through "e"?)
testCheck = "aM5b" Like "a[L-P]#[!c-e]"
' The following statement returns True (does "BAT123khg" begin with a
'    "B", followed by any single character, followed by a "T", and end
'    with zero or more characters of any type?)
testCheck = "BAT123khg" Like "B?T*"
' The following statement returns False (does "CAT123khg"?) begin with
'    a "B", followed by any single character, followed by a "T", and
'    end with zero or more characters of any type?)
testCheck = "CAT123khg" Like "B?T*"
```

Voir aussi

Tâches

Comment : faire correspondre une chaîne à un modèle

Référence

Opérateurs de comparaison (Visual Basic)

InStr, fonction (Visual Basic)

Priorité des opérateurs en Visual Basic

Opérateurs répertoriés par fonctionnalité

Option Compare, instruction

StrComp, fonction (Visual Basic)

Concepts

Opérateurs et expressions en Visual Basic

Mod, opérateur (Visual Basic)

Effectue la division de deux nombres et retourne seulement le reste.

```
number1 Mod number2
```

Éléments

number1

Requis. Toute expression numérique.

number2

Requis. Toute expression numérique.

Types pris en charge

Tous les types numériques, y compris les types non signés et à virgule flottante, ainsi que **Decimal**.

Résultat

Le résultat représente le reste après la division de *number1* par *number2*. Par exemple, l'expression `14 Mod 4` prend la valeur 2.

Notes

Si *number1* ou *number2* est une valeur à virgule flottante, le reste à virgule flottante de la division est retourné. Le type de données du résultat est le plus petit type de données qui peut contenir toutes les valeurs possibles résultant de la division avec les types de données de *number1* et *number2*.

Si *number1* ou *number2* correspond à Nothing, elle est considérée comme un zéro.

Les opérateurs connexes incluent les éléments suivants :

- \, opérateur retourne le quotient entier d'une division. Par exemple, l'expression `14 \ 4` prend la valeur 3.
- /, opérateur (Visual Basic) retourne le quotient complet, y compris le reste, en tant que nombre à virgule flottante. Par exemple, l'expression `14 / 4` prend la valeur 3,5.

Essai de division par zéro

Si *number2* correspond à zéro, le comportement de l'opérateur **Mod** dépend du type de données des opérandes. Une division intégrale lève une exception DivideByZeroException. Une division de nombres à virgule flottante retourne NaN.

Formule équivalente

L'expression `a Mod b` est équivalente à l'une ou l'autre des formules suivantes :

```
a - (b * (a \ b))
```

```
a - (b * Fix(a / b))
```

Imprécision des nombres à virgule flottante

Lorsque vous utilisez des nombres à virgule flottante, pensez qu'ils n'ont pas toujours de représentation précise dans la mémoire. Certaines opérations pourraient avoir des résultats inattendus, comme la comparaison de valeur et l'opérateur **Mod**. Pour plus d'informations, consultez Dépannage des types de données.

Surcharge

L'opérateur **Mod** *peut être surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement. Si votre code applique **Mod** sur l'instance d'une classe ou d'une structure qui inclut une telle surcharge, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **Mod** pour diviser deux nombres et retourner seulement le reste. Si l'un des nombres est à virgule flottante, le résultat est un nombre à virgule flottante représentant le reste.

VB

```
Dim testResult As Double
testResult = 10 Mod 5
testResult = 10 Mod 3
testResult = 12 Mod 4.3
testResult = 12.6 Mod 5
testResult = 47.9 Mod 9.35
```

Les expressions dans les exemples précédents retournent des valeurs de 0 ; 1 ; 3,4 ; 2,6 et 1,15.

L'exemple suivant montre l'imprécision potentielle des opérandes à virgule flottante. Dans la première instruction, les opérandes sont **Double** et 0,2 représente une fraction binaire se répétant à l'infini avec une valeur stockée de 0,20000000000000001. Dans la deuxième instruction, le caractère de type de littéral `D` force les deux opérandes sur la valeur **Decimal** et 0,2 a une représentation précise.

VB

```
firstResult = 2.0 Mod 0.2
' Double operation returns 0.2, not 0.
secondResult = 2D Mod 0.2D
' Decimal operation returns 0.
```

Voir aussi

Tâches

Dépannage des types de données

Référence

Opérateurs arithmétiques (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
Fonctions Int, Fix (Visual Basic)
\, opérateur

Concepts

Opérateurs arithmétiques en Visual Basic

Not, opérateur (Visual Basic)

Effectue une négation logique sur une expression **Boolean** ou une négation de bits sur une expression numérique.

```
result = Not expression
```

Éléments

result

Requis. Toute expression **Boolean** ou numérique.

expression

Requis. Toute expression **Boolean** ou numérique.

Notes

Pour les expressions **Boolean**, le tableau suivant illustre la manière dont l'argument *result* est déterminé :

Si *expression* a la valeur	**La valeur de *result* est**
True	**False**
False	**True**

Pour les expressions numériques, l'opérateur **Not** inverse les valeurs de bits de toute expression numérique et définit le bit correspondant dans *result* d'après le tableau suivant :

Si le bit dans *expression* a la valeur	**Le bit dans *result* a la valeur**
1	0
0	1

Remarque

Compte tenu que les opérateurs logiques et de bits ont une priorité inférieure à celle des opérateurs arithmétiques et relationnels, toutes les opérations au niveau du bit doivent être mises entre parenthèses afin de garantir une exécution précise.

Types de données

Pour une négation Boolean, le type de données du résultat est **Boolean**. Pour une négation au niveau du bit, le type de données de résultat est le même que celui de *expression*. Toutefois, si l'expression est **Decimal**, le résultat est **Long**.

Surcharge

L'opérateur **Not** peut être *surchargé*, ce qui signifie qu'une classe ou structure peut redéfinir son comportement lorsque son opérande a le type de cette classe ou structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **Not** pour effectuer une négation logique sur une expression **Boolean**. Le résultat est une valeur **Boolean** qui représente l'inverse de la valeur de l'expression.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck As Boolean
firstCheck = Not (a > b)
secondCheck = Not (b > a)
```

L'exemple précédent produit respectivement les résultats suivants : **False** et **True**.

L'exemple suivant utilise l'opérateur **Not** pour effectuer une négation logique sur les bits individuels d'une expression numérique. Le bit dans le modèle de résultat a la valeur de l'inverse du bit correspondant dans le modèle d'opérande, y compris le bit de signe.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (Not a)
secondPattern = (Not b)
thirdPattern = (Not c)
```

L'exemple précédent produit respectivement les résultats suivants : -11, -9, et -7.

Voir aussi

Référence

Opérateurs logiques/de bits

Priorité des opérateurs en Visual Basic

Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

Or, opérateur (Visual Basic)

Effectue une disjonction logique sur deux expressions **Boolean** ou une disjonction d'opérations de bits sur deux expressions numériques.

```
result = expression1 Or expression2
```

Éléments

result

Requis. Toute expression **Boolean** ou numérique. Pour la comparaison **Boolean**, *result* est la disjonction logique inclusive de deux valeurs **Boolean**. Pour les opérations au niveau du bit, *result* est une valeur numérique qui représente la disjonction de bits inclusive de deux modèles binaires numériques.

expression1

Requis. Toute expression **Boolean** ou numérique.

expression2

Requis. Toute expression **Boolean** ou numérique.

Notes

Pour la comparaison **Boolean**, *result* est **False** si et uniquement si *expression1* et *expression2* ont la valeur **False**. Le tableau suivant illustre la manière dont *result* est déterminé.

Si ***expression1*** a la valeur	Et ***expression2*** a la valeur	La valeur de ***result*** est
True	**True**	**True**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

Remarque

Dans une comparaison **Boolean**, l'opérateur **Or** évalue toujours les deux expressions qui pourraient inclure des appels de pr océdure. L'opérateur OrElse effectue un *court-circuit* ce qui signifie que si *expression1* a la valeur **True**, alors *expression2* n'es t pas évalué.

Pour les opérations au niveau du bit, l'opérateur **Or** effectue une comparaison de bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après le tableau suivant :

Si le bit dans ***expression1*** a la valeur	Et si le bit dans ***expression2*** a la valeur	Le bit dans ***result*** a la valeur
1	1	1
1	0	1
0	1	1
0	0	0

Remarque

Compte tenu que les opérateurs logiques et de bits ont une priorité inférieure à celle des opérateurs arithmétiques et relatio nnels, toutes les opérations au niveau du bit doivent être mises entre parenthèses afin de garantir une exécution précise.

Types de données

Si les opérandes se composent d'une expression **Boolean** et d'une expression numérique, Visual Basic convertit l'expression **Boolean** en une valeur numérique (–1 pour **True** et 0 pour **False**) et exécute une opération au niveau du bit.

Pour une comparaison **Boolean**, le type de données du résultat est **Boolean**. Pour une comparaison de bits, le type de données du résultat est un type numérique approprié pour les types de données de *expression1* et *expression2*. Consultez la table « Relational and Bitwise Comparisons » dans Types de données des résultats d'opérateur.

Surcharge

L'opérateur **Or** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **Or** pour effectuer une disjonction logique inclusive sur deux expressions. Le résultat est une valeur **Boolean** qui indique si l'une des deux expressions est **True**.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b Or b > c
secondCheck = b > a Or b > c
thirdCheck = b > a Or c > b
```

L'exemple précédent produit respectivement les résultats suivants : **True**, **True** et **False**.

L'exemple suivant utilise l'opérateur **Or** pour effectuer une exclusion logique inclusive sur les bits individuels de deux expressions numériques. Le bit du modèle de résultat est défini si un des bits correspondants dans les opérandes a la valeur 1.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (a Or b)
secondPattern = (a Or c)
thirdPattern = (b Or c)
```

L'exemple précédent produit respectivement les résultats suivants : 10, 14 et 14.

Voir aussi

Référence

Opérateurs logiques/de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
OrElse, opérateur

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

OrElse, opérateur

Effectue une disjonction logique inclusive "court-circuitante" sur deux expressions.

```
result = expression1 OrElse expression2
```

Éléments

result

Requis. Toute expression **Boolean**.

expression1

Requis. Toute expression **Boolean**.

expression2

Requis. Toute expression **Boolean**.

Notes

Une opération logique est dite de *court-circuit* si le code compilé peut ignorer l'évaluation d'une expression en fonction du résultat d'une autre expression. Si le résultat de la première expression évaluée détermine le résultat final de l'opération, il n'y a pas besoin d'évaluer la seconde expression, car elle ne peut pas changer le résultat final. Un court circuit peut améliorer les performances si l'expression ignorée est complexe, ou si elle implique des appels de procédure.

Si une ou les deux expressions ont la valeur **True**, *result* a la valeur **True**. Le tableau suivant illustre la manière dont *result* est déterminé.

Si ***expression1*** a la valeur	Et ***expression2*** a la valeur	La valeur de ***result*** est
True	(non évaluée)	**True**
False	**True**	**True**
False	**False**	**False**

Types de données

L'opérateur **OrElse** est défini uniquement pour Boolean, type de données (Visual Basic). Visual Basic convertit chaque opérande si nécessaire en **Boolean** et exécute l'opération entière en **Boolean**. Si vous assignez le résultat à un type numérique, Visual Basic le convertit de **Boolean** en ce type. Cette conversion pourrait avoir comme conséquence un comportement inattendu. Par exemple, `5 OrElse 12` donne `-1` lorsqu'il est converti en **Integer**.

Surcharge

Les opérateurs Or (Visual Basic) et IsTrue peuvent être *surchargés*, ce qui signifie qu'une classe ou une structure peut redéfinir leur comportement lorsqu'un opérande a le type de cette classe ou de cette structure. La surcharge des opérateurs **Or** et **IsTrue** affecte le comportement de l'opérateur **OrElse**. Si votre code utilise **OrElse** sur une classe ou une structure qui surcharge **Or** et **IsTrue**, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

L'exemple suivant utilise l'opérateur **OrElse** pour effectuer une disjonction logique sur deux expressions. Le résultat est une valeur **Boolean** qui indique si l'une des deux expressions est true. Si la première expression est **True**, la seconde n'est pas évaluée.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b OrElse b > c
secondCheck = b > a OrElse b > c
```

```
thirdCheck = b > a OrElse c > b
```

L'exemple précédent produit respectivement les résultats suivants : **True**, **True** et **False**. Dans le calcul de `firstCheck`, la seconde expression n'est pas évaluée parce que la première a déjà la valeur **True**. Toutefois, la seconde expression est évaluée dans le calcul de `secondCheck`.

L'exemple suivant affiche une instruction **If**...**Then** qui contient deux appels de procédure. Si le premier appel retourne **True**, la seconde procédure n'est pas appelée. Si la seconde procédure effectue des tâches importantes qui doivent toujours être effectuées lorsque cette section de code est exécutée, des résultats inattendus pourraient se produire.

VB

```
If testFunction(5) = True OrElse otherFunction(4) = True Then
    ' If testFunction(5) is True, otherFunction(4) is not called.
    ' Insert code to be executed.
End If
```

Voir aussi

Référence

Opérateurs logiques/de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
Or, opérateur (Visual Basic)
IsTrue, opérateur

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

TypeOf, opérateur (Visual Basic)

Compare une variable référence d'objet à un type de données.

```
result = TypeOf objectexpression Is typename
```

Éléments

result

Retourné. Valeur **Boolean**.

objectexpression

Requis. Toute expression ayant pour résultat un type référence.

typename

Requis. Tout nom de type de données.

Notes

L'opérateur **TypeOf** détermine si le type d'exécution de *objectexpression* est compatible avec *typename*. La compatibilité dépend de la catégorie du type de *typename*. Le tableau suivant illustre la manière dont la compatibilité est déterminée.

Catégorie du type de *typename*	**Critère de compatibilité**
Classe	*objectexpression* est de type *typename* ou hérite de *typename*
Structure	*objectexpression* est de type *typename*
Interface	*objectexpression* implémente *typename* ou hérite d'une classe qui implémente *typename*

Si le type d'exécution de *objectexpression* satisfait le critère de compatibilité, *result* est **True**. Sinon, *result* est **False**.

TypeOf est toujours utilisé avec le mot clé **Is** pour construire une expression **TypeOf**...**Is**.

Exemple

L'exemple suivant utilise des expressions **TypeOf**...**Is** pour tester la compatibilité du type de deux variables référence d'objet avec divers types de données.

VB

```
Dim refInteger As Object = 2
MsgBox("TypeOf Object[Integer] Is Integer? " & TypeOf refInteger Is Integer)
MsgBox("TypeOf Object[Integer] Is Double? " & TypeOf refInteger Is Double)
Dim refForm As Object = New System.Windows.Forms.Form
MsgBox("TypeOf Object[Form] Is Form? " & TypeOf refForm Is System.Windows.Forms.Form)
MsgBox("TypeOf Object[Form] Is Label? " & TypeOf refForm Is System.Windows.Forms.Label)
MsgBox("TypeOf Object[Form] Is Control? " & TypeOf refForm Is System.Windows.Forms.Control)
MsgBox("TypeOf Object[Form] Is IComponent? " & TypeOf refForm Is System.ComponentModel.ICom
ponent)
```

Le `refInteger` variable a un type d'exécution de **Integer**. Il est compatible avec **Integer**, mais pas avec **Double**. Le `refForm` variable a un type d'exécution de Form. Il est compatible avec **Form**, car il s'agit de son type, avec Control parce que **Form** hérite de **Control**, et avec IComponent parce que **Form** hérite de Componentqui implémente **IComponent**. Toutefois, `refForm` n'est pas compatible avec Label.

Voir aussi

Référence

Is, opérateur (Visual Basic)
IsNot, opérateur
Priorité des opérateurs en Visual Basic

Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs de comparaison en Visual Basic

Opérateurs et expressions en Visual Basic

Xor, opérateur (Visual Basic)

Effectue une exclusion logique sur deux expressions **Boolean** ou une exclusion d'opérations de bits sur deux expressions numériques.

```
result = expression1 Xor expression2
```

Éléments

result

Requis. Toute variable **Boolean** ou numérique. Pour la comparaison Boolean, *result* est l'exclusion logique (disjonction logique exclusive) de deux valeurs **Boolean**. Pour les opérations au niveau du bit, *result* est une valeur numérique représentant l'exclusion d'opérations de bits (disjonction d'opérations de bits exclusive) de deux modèles binaires numériques.

expression1

Requis. Toute expression **Boolean** ou numérique.

expression2

Requis. Toute expression **Boolean** ou numérique.

Notes

Pour la comparaison Boolean, *result* a la valeur **True** si et uniquement si exactement *expression1* ou *expression2* a la valeur **True**, c'est-à-dire, si et uniquement si *expression1* et *expression2* ont les valeurs opposées à la valeur **Boolean**. Le tableau suivant illustre la manière dont *result* est déterminé.

Si *expression1* a la valeur	**Et *expression2* a la valeur**	**La valeur de *result* est**
True	**True**	**False**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

Remarque

Dans la comparaison Boolean, l'opérateur **Xor** évalue toujours les deux expressions qui pourraient inclure des appels de proc édure. Il n'existe pas d'équivalent court-circuitant de **Xor**, étant donné que le résultat dépend toujours des deux opérandes. P our obtenir des informations sur les opérateurs logiques *court-circuitant*, consultez AndAlso, opérateur et OrElse, opérateur.

Pour les opérations au niveau du bit, l'opérateur **Xor** effectue une comparaison de bits de position identique dans deux expressions numériques et définit le bit correspondant dans *result* d'après le tableau suivant :

Si le bit dans *expression1* est	**Et si le bit dans *expression2* est**	**Le bit dans *result* est**
1	1	0
1	0	1
0	1	1
0	0	0

Remarque

Compte tenu que les opérateurs logiques et de bits ont une priorité inférieure à celle des opérateurs arithmétiques et relationnels, toutes les opérations au niveau du bit doivent être mises entre parenthèses afin de garantir une exécution précise.

Par exemple, 5 **Xor** 3 équivaut à 6. Pour connaître la raison de cette équivalence, convertissez 5 et 3 en leurs représentations binaire, 101 et 011. Puis, utilisez la table précédente pour déterminer que 101 Xor 011 est égal à 110, représentation binaire du nombre décimal 6.

Types de données

Si les opérandes se composent d'une expression **Boolean** et d'une expression numérique, Visual Basic convertit l'expression **Boolean** en une valeur numérique (-1 pour **True** et 0 pour **False**) et exécute une opération au niveau du bit.

Pour une comparaison **Boolean**, le type de données du résultat est **Boolean**. Pour une comparaison de bits, le type de données du résultat est un type numérique approprié pour les types de données de *expression1* et *expression2*. Consultez le tableau « Relational and Bitwise Comparisons » dans Types de données des résultats d'opérateur.

Surcharge

L'opérateur **Xor** peut être *surchargé*, ce qui signifie qu'une classe ou une structure peut redéfinir son comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise cet opérateur sur une telle classe ou structure, assurez-vous que vous comprenez son comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Exemple

Cet exemple utilise l'opérateur **Xor** pour effectuer une exclusion logique (disjonction logique exclusive) sur deux expressions. Le résultat est une valeur **Boolean** qui indique si une seule des expressions a la valeur **True**.

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b Xor b > c
secondCheck = b > a Xor b > c
thirdCheck = b > a Xor c > b
```

L'exemple précédent produit respectivement les résultats suivants : **False**, **True** et **False**.

Cet exemple utilise l'opérateur **Xor** pour effectuer une exclusion logique (disjonction logique exclusive) sur les bits de deux expressions numériques. Le bit du modèle de résultat est défini si un seul des bits correspondants dans les opérandes a la valeur 1.

VB

```
Dim a As Integer = 10 ' 1010 in binary
Dim b As Integer = 8  ' 1000 in binary
Dim c As Integer = 6  ' 0110 in binary
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (a Xor b)  '  2, 0010 in binary
secondPattern = (a Xor c) ' 12, 1100 in binary
thirdPattern = (b Xor c)  ' 14, 1110 in binary
```

L'exemple précédent produit respectivement les résultats suivants : 2, 12 et 14,.

Voir aussi

Référence

Opérateurs logiques/de bits
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

Types de données des résultats d'opérateur

Visual Basic détermine le type de données de résultat d'une opération à partir des types de données des opérandes. Dans certains cas, il peut s'agir d'un type de données disposant d'une plage plus grande que celle d'un opérande.

Plages des types de données

Les plages des types de données pertinents, dans l'ordre croissant de taille, sont les suivantes :

- Boolean — deux valeurs possibles
- SByte, Byte — 256 valeurs intégrales possibles
- Short, UShort — 65 536 (6,5... E+4) valeurs intégrales possibles
- Integer, UInteger — 4 294 967 296 (4,2... E+9) valeurs intégrales possibles
- Long, ULong — 18 446 744 073 709 551 615 (1,8... E+9) valeurs intégrales possibles
- Decimal — 1,5... E+29 valeurs intégrales possibles, la plage maximale est 7,9... E+28 (valeur absolue)
- Single — la plage maximale est 3,4 ... E+38 (valeur absolue)
- Double — la plage maximale est 1,7 ... E+308 (valeur absolue)

Pour plus d'informations sur les types de données Visual Basic, consultez Liste des types de données (Visual Basic).

Si un opérande prend la valeur Nothing, les opérateurs arithmétiques Visual Basic la considèrent comme étant égale à zéro.

Arithmétique décimale

Notez que le type de données Decimal n'est ni à virgule flottante, ni entier.

Si un opérande d'une opération **+**, **–**, *****, **/**, ou **Mod** est de type **Decimal** et que l'autre n'est pas **Single** ou **Double**, Visual Basic étend l'autre opérande à **Decimal**. Il effectue l'opération en **Decimal** et le type de données de résultat est **Decimal**.

Arithmétique à virgule flottante

Visual Basic exécute la plupart des opérations arithmétiques à virgule flottante en Double, qui est le type de données le plus efficace pour ce genre d'opérations. Toutefois, si un opérande est Single et que l'autre n'est pas **Double**, Visual Basic effectue l'opération en **Single**. Il étend si nécessaire chaque opérande au type de données approprié avant l'opération, et le résultat est du même type de données.

Opérateurs / et ^

L'opérateur **/** est défini uniquement pour les types de données Decimal, Single et Double. Visual Basic étend si nécessaire chaque opérande au type de données approprié avant l'opération, et le résultat est du même type de données.

Le tableau suivant présente les types de données de résultat pour l'opérateur **/**. Notez que ce tableau est symétrique ; pour une combinaison de types de données d'opérande spécifiée, le type de données de résultat est le même, quel que soit l'ordre des opérandes.

	Decimal	**Single**	**Double**	Tout type d'entier
Decimal	Decimal	Single	Double	Double
Single	Single	Single	Double	Double
Double	Double	Double	Double	Double
Tout type d'entier	Double	Double	Double	Double

L'opérateur **^** est défini uniquement pour le type de données **Double**. Visual Basic étend si nécessaire chaque opérande à **Double** avant l'opération, et le type de données de résultat est toujours **Double**.

Arithmétique sur les entiers

Le type de données de résultat d'une opération sur des entiers dépend des types de données des opérandes. En général, Visual Basic utilise les stratégies suivantes pour déterminer le type de données de résultat :

- Si les deux opérandes d'un opérateur binaire ont le même type de données, le résultat sera également de ce type de données. **Boolean** fait exception ; il est forcé en **Short**.
- Si un opérande non signé est associé à un opérande signé, le résultat est d'un type signé assorti d'une plage au moins aussi grande que l'un des opérandes.
- Sinon, le résultat prend généralement le plus grand des deux types de données d'opérande.

Notez que le type de données de résultat peut ne pas être le même que celui des opérandes.

Remarque
En effet, le type de données de résultat n'est pas toujours assez grand pour contenir toutes les valeurs possibles résultant de l'opération. Une exception OverflowException peut se produire si la valeur est trop grande pour le type de données de résult at.

Opérateurs unaires + et -

Le tableau suivant présente les types de données de résultat pour les deux opérateurs unaires, **+** et **–**.

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
+ unaire	Short	SByte	Byte	Short	UShort	Integer	UInteger	Long	ULong
– unaire	Short	SByte	Short	Short	Integer	Integer	Long	Long	Decimal

Opérateurs << et >>

Le tableau suivant présente les types de données de résultat pour les deux opérateurs de décalage de bits, **<<** et **>>**. Visual Basic traite chaque opérateur de décalage de bits comme un opérateur unaire sur son opérande gauche (modèle binaire à décaler).

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
<<, **>>**	Short	SByte	Byte	Short	UShort	Integer	UInteger	Long	ULong

Si l'opérande gauche est de type **Decimal**, **Single**, **Double** ou **String**, Visual Basic essaie de le convertir en **Long** avant l'opération, et le type de données de résultat est **Long**. L'opérande droit (le nombre de positions binaires à décaler) doit être **Integer** ou d'un type qui s'étend à **Integer**.

Opérateurs binaires + et –, et opérateurs * et Mod

Le tableau suivant présente les types de données de résultat pour les opérateurs binaires **+** et **–** et les opérateurs ***** et **Mod**. Notez que ce tableau est symétrique ; pour une combinaison de types de données d'opérande spécifiée, le type de données de résultat est le même, quel que soit l'ordre des opérandes.

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
Boolean	Short	SByte	Short	Short	Integer	Integer	Long	Long	Decimal
SByte	SByte	SByte	Short	Short	Integer	Integer	Long	Long	Decimal
Byte	Short	Short	Byte	Short	UShort	Integer	UInteger	Long	ULong
Short	Short	Short	Short	Short	Integer	Integer	Long	Long	Decimal
UShort	Integer	Integer	UShort	Integer	UShort	Integer	UInteger	Long	ULong

Integer	Integer	Integer	Integer	Integer	Integer	Integer	Long	Long	Decimal
UInteger	Long	Long	UInteger	Long	UInteger	Long	UInteger	Long	ULong
Long	Long	Long	Long	Long	Long	Long	Long	Long	Decimal
ULong	Decimal	Decimal	ULong	Decimal	ULong	Decimal	ULong	Decimal	ULong

\ opérateur

Le tableau suivant présente les types de données de résultat pour l'opérateur ****. Notez que ce tableau est symétrique ; pour une combinaison de types de données d'opérande spécifiée, le type de données de résultat est le même, quel que soit l'ordre des opérandes.

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
Boolean	Short	SByte	Short	Short	Integer	Integer	Long	Long	Long
SByte	SByte	SByte	Short	Short	Integer	Integer	Long	Long	Long
Byte	Short	Short	Byte	Short	UShort	Integer	UInteger	Long	ULong
Short	Short	Short	Short	Short	Integer	Integer	Long	Long	Long
UShort	Integer	Integer	UShort	Integer	UShort	Integer	UInteger	Long	ULong
Integer	Integer	Integer	Integer	Integer	Integer	Integer	Long	Long	Long
UInteger	Long	Long	UInteger	Long	UInteger	Long	UInteger	Long	ULong
Long	Long	Long	Long	Long	Long	Long	Long	Long	Long
ULong	Long	Long	ULong	Long	ULong	Long	ULong	Long	ULong

Si l'un des opérandes de l'opérateur **** est Decimal, Single ou Double, Visual Basic essaie de le convertir en Long avant l'opération, et le type de données de résultat est **Long**.

Comparaison de bits et comparaison relationnelle

Le type de données de résultat d'une opération relationnelle (**=**, **<>**, **<**, **>**, **<=**, **>=**) est toujours **Boolean**Boolean, type de données (Visual Basic). Il en va de même pour les opérations logiques (**And**, **AndAlso**, **Not**, **Or**, **OrElse**, **Xor**) sur les opérandes **Boolean**.

Le type de données de résultat d'une opération logique de bits dépend des types de données des opérandes. Notez que **AndAlso** et **OrElse** sont définis uniquement pour **Boolean** et que Visual Basic convertit si nécessaire chaque opérande en **Boolean** avant d'effectuer l'opération.

Opérateurs =, <>, <, >, <= et >=

Si les deux opérandes sont **Boolean**, Visual Basic considère que la valeur **True** est inférieure à **False**. Si un type numérique est comparé à un **String**, Visual Basic essaie de convertir le **String** en **Double** avant l'opération. Une comparaison avec un opérande de type **Char** ou **Date** ne peut être établie qu'avec un autre opérande du même type de données. Le type de données de résultat est toujours **Boolean**.

Opérateur de bits NOT

Le tableau suivant présente les types de données de résultat pour l'opérateur de bits **Not**.

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**

Not	Boolean	SByte	Byte	Short	UShort	Integer	UInteger	Long	ULong

Si l'opérande est de type **Decimal**, **Single**, **Double** ou **String**, Visual Basic essaie de le convertir en **Long** avant l'opération, et le type de données de résultat est **Long**.

Opérateurs de bits AND, OR et XOR

Le tableau suivant présente les types de données de résultat pour les opérateurs de bits **And**, **Or** et **Xor**. Notez que ce tableau est symétrique ; pour une combinaison de types de données d'opérande spécifiée, le type de données de résultat est le même, quel que soit l'ordre des opérandes.

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
Boolean	Boolean	SByte	Short	Short	Integer	Integer	Long	Long	Long
SByte	SByte	SByte	Short	Short	Integer	Integer	Long	Long	Long
Byte	Short	Short	Byte	Short	UShort	Integer	UInteger	Long	ULong
Short	Short	Short	Short	Short	Integer	Integer	Long	Long	Long
UShort	Integer	Integer	UShort	Integer	UShort	Integer	UInteger	Long	ULong
Integer	Integer	Integer	Integer	Integer	Integer	Integer	Long	Long	Long
UInteger	Long	Long	UInteger	Long	UInteger	Long	UInteger	Long	ULong
Long	Long	Long	Long	Long	Long	Long	Long	Long	Long
ULong	Long	Long	ULong	Long	ULong	Long	ULong	Long	ULong

Si un opérande est **Decimal**, **Single**, **Double** ou **String**, Visual Basic essaie de le convertir en **Long** avant l'opération, et le type de données de résultat est le même que si cet opérande était déjà de type **Long**.

Opérateurs divers

L'opérateur **&** est défini uniquement pour la concaténation d'opérandes **String**. Visual Basic convertit chaque opérande si nécessaire en **String** avant l'opération, et le type de données de résultat est toujours **String**. Pour les besoins de l'opérateur **&**, toutes les conversions en **String** sont considérées comme étant étendues, même si **Option Strict** est **On**.

Les opérateurs **Is** et **IsNot** exigent que les deux opérandes soient d'un type référence. L'expression **TypeOf**...**Is** exige que le premier opérande soit d'un type référence et que le second opérande représente le nom d'un type de données. Dans tous ces cas, le type de données de résultat est **Boolean**.

L'opérateur **Like** est défini uniquement pour les critères spéciaux d'opérandes **String**. Visual Basic essaie si nécessaire de convertir chaque opérande en **String** avant l'opération. Le type de données de résultat est toujours **Boolean**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
Opérateurs arithmétiques (Visual Basic)
Opérateurs de comparaison (Visual Basic)
Option Strict, instruction

Concepts

Opérateurs et expressions en Visual Basic
Opérateurs arithmétiques en Visual Basic
Opérateurs de comparaison en Visual Basic

Autres ressources

Opérateurs (Visual Basic)

Opérateurs arithmétiques (Visual Basic)

Les opérateurs arithmétiques définis dans Visual Basic sont les suivants :

^, opérateur

*, opérateur

/, opérateur

\, opérateur

Mod, opérateur

+ opérateur (unaire et binaire)

- opérateur (unaire et binaire)

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

Concepts

Opérateurs arithmétiques en Visual Basic

Opérateurs d'assignation

Les opérateurs d'assignation définis dans Visual Basic sont les suivants.

=, opérateur

^=, opérateur

*=, opérateur

/=, opérateur

\=, opérateur

+=, opérateur

-=, opérateur

<<=, opérateur

>>=, opérateur

&=, opérateur

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

Opérateurs répertoriés par fonctionnalité

Concepts

Instructions d'assignation

Opérateurs de décalage de bits

Les opérateurs de décalage de bits définis dans Visual Basic sont les suivants :

<< , opérateur

>> , opérateur

Voir aussi

Référence

Opérateurs répertoriés par fonctionnalité

Opérateurs de comparaison (Visual Basic)

Les opérateurs de comparaison définis dans Visual Basic sont les suivants.

Opérateur **<**

Opérateur **<=**

Opérateur **>**

Opérateur **>=**

Opérateur **=**

Opérateur **<>**

Is, opérateur (Visual Basic)

IsNot, opérateur

Like, opérateur

Ces opérateurs comparent deux expressions pour déterminer si elles sont égales et, dans le cas contraire, identifier leurs différences. **Is**, **IsNot** et **Like** sont abordés en détail à différentes pages de l'aide. Cette page aborde en détail les opérateurs de comparaison relationnels.

```
result = expression1 comparisonoperator expression2
result = object1 [Is | IsNot] object2
result = string Like pattern
```

Éléments
result

Requis. Valeur **Boolean** représentant le résultat de la comparaison.

expression

Requis. Toute expression.

comparisonoperator

Requis. Tout opérateur de comparaison relationnel.

object1, *object2*

Requis. Tout nom d'objet de référence.

string

Requis. Toute expression **String**.

pattern

Requis. Toute expression **String** ou plage de caractères.

Notes

Le tableau suivant contient une liste des opérateurs de comparaison relationnels et des conditions qui déterminent si *result* a la valeur **True** ou **False**.

Opérateur	**True si**	**False si**
< (inférieur à)	*expression1* < *expression2*	*expression1* >= *expression2*
<= (inférieur ou égal à)	*expression1* <= *expression2*	*expression1* > *expression2*
> (supérieur à)	*expression1* > *expression2*	*expression1* <= *expression2*

>= (supérieur ou égal à)	*expression1* >= *expression2*	*expression1* < *expression2*
= (égal à)	*expression1* = *expression2*	*expression1* <> *expression2*
<> (différent de)	*expression1* <> *expression2*	*expression1* = *expression2*

Remarque

Le =, opérateur (Visual Basic) est également utilisé comme opérateur d'assignation.

L'opérateur **Is**, l'opérateur **IsNot** et l'opérateur **Like** ont des fonctionnalités de comparaison spécifiques qui sont en désaccord avec les opérateurs dans le tableau précédent.

Comparaison de nombres

Lorsqu'une expression de type **Single** est comparée à une expression de type **Double**, l'expression **Single** est convertie en **Double**. Ce comportement est l'inverse de celui trouvé dans Visual Basic 6.

De la même façon, lorsque vous comparez une expression de type **Decimal** à une expression de type **Single** ou **Double**, l'expression **Decimal** est convertie en **Single** ou **Double**. Pour les expressions **Decimal**, toute valeur fractionnaire inférieure à *1E-28* risque d'être perdue. Du fait de cette perte de valeur fractionnaire, deux valeurs peuvent être considérées comme égales alors qu'elles ne le sont pas. C'est pourquoi le plus grand soin est nécessaire lors de l'utilisation de l'égalité (**=**) pour comparer deux variables à virgule flottante. Il vaut mieux s'assurer que la valeur absolue de la différence entre les deux nombres est tout au plus une petite tolérance acceptable.

Imprécision des nombres à virgule flottante

Lorsque vous utilisez des nombres à virgule flottante, pensez qu'ils n'ont pas toujours de représentation précise dans la mémoire. Certaines opérations pourraient avoir des résultats inattendus, comme la comparaison de valeur et l'opérateur Mod, opérateur (Visual Basic). Pour plus d'informations, consultez Dépannage des types de données.

Comparaison de chaînes

Lorsque vous comparez des chaînes, les expressions de chaîne sont évaluées selon leur ordre de tri alphabétique qui dépend du paramètre **Option Compare**.

Option Compare Binary donne lieu à des comparaisons de chaînes basées sur un ordre de tri dérivé de la représentation binaire interne des caractères. L'ordre de tri est déterminé par la page de codes. L'exemple suivant montre un ordre de tri binaire typique.

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

Option Compare Text fournit des comparaisons de chaînes basées sur un ordre de tri de texte qui ne tient pas compte de la casse et qui est déterminé par les paramètres régionaux de votre application. Lorsque vous définissez **Option Compare Text** et triez les caractères dans l'exemple précédent, l'ordre de tri de texte suivant s'applique :

```
(A=a) < (À= à) < (B=b) < (E=e) < (Ê= ê) < (Ø = ø) < (Z=z)
```

Dépendance des paramètres régionaux

Lorsque vous définissez **Option Compare Text**, le résultat d'une comparaison de chaînes peut dépendre des paramètres régionaux définis pour l'application. Deux caractères peuvent être équivalents selon certains paramètres régionaux, mais pas selon d'autres. Si vous utilisez une comparaison de chaînes pour prendre des décisions importantes, par exemple pour décider si vous pouvez accepter ou non une tentative de connexion, vous devez être conscient du critère de diffusion des paramètres régionaux. Pensez à définir **Option Compare Binary** ou appeler le StrComp, fonction (Visual Basic) qui prend les paramètres régionaux en considération.

Programmation sans type avec les opérateurs de comparaison relationnels

L'utilisation d'opérateurs de comparaison relationnels avec des expressions **Object** est interdite sous **Option Strict On**. Lorsque **Option Strict** est **Off** et *expression1* ou *expression2* est une expression **Object**, les types d'exécution déterminent la façon dont ils sont comparés. Le tableau suivant montre la façon dont les expressions sont comparées et ce qui résulte de la comparaison, en fonction du type d'exécution des opérandes :

Si les opérandes sont :	**La comparaison est**
Les deux **String**	Comparaison de tri à partir des caractéristiques de tri des chaînes.

Les deux numériques	Objets convertis en **Double**, comparaison numérique.
Une valeur numérique et une valeur **String**	La valeur **String** est convertie en **Double** et la comparaison numérique est effectuée. Si **String** ne peut être converti en **Double**, une exception InvalidCastException est levée.
L'une ou les deux sont des types référence autres que **String**.	Un **InvalidCastException** est levé.

Les comparaisons numériques traitent **Nothing** comme la valeur 0. Les comparaisons de chaînes traitent **Nothing** comme **""** (chaîne vide).

Surcharge

Les opérateurs de comparaison relationnels (**<**, **<=**, **>**, **>=**, **=**, **<>**) peuvent être *surchargés*, ce qui signifie qu'une classe ou une structure peut redéfinir leur comportement lorsqu'un opérande a le type de cette classe ou de cette structure. Si votre code utilise l'un de ces opérateurs sur une telle classe ou structure, assurez-vous que vous comprenez le comportement redéfini. Pour plus d'informations, consultez Procédures d'opérateur.

Notez que le =, opérateur (Visual Basic) peut être surchargé uniquement en tant qu'opérateur de comparaison relationnel, pas en tant qu'opérateur d'assignation.

Exemple

L'exemple suivant montre les différentes utilisations des opérateurs de comparaison relationnels dont vous vous servez pour comparer des expressions. Les opérateurs de comparaison relationnels retournent un résultat **Boolean** qui indique si l'expression spécifiée a la valeur **True**. Lorsque vous appliquez les opérateurs **>** et **<** à des chaînes, la comparaison est effectuée à l'aide de l'ordre de tri alphabétique normal des chaînes. Cet ordre peut dépendre de vos paramètres régionaux. Le fait que le tri respecte la casse ou non dépend du paramètre Option Compare.

VB

```
Dim testResult As Boolean
testResult = 45 < 35
testResult = 45 = 45
testResult = 4 <> 3
testResult = "5" > "4444"
```

Dans l'exemple précédent, la première comparaison retourne **False** et les comparaisons restantes retournent **True**.

Voir aussi

Tâches

Dépannage des types de données

Référence

=, opérateur (Visual Basic)
Priorité des opérateurs en Visual Basic
Opérateurs répertoriés par fonctionnalité
InvalidCastException

Concepts

Opérateurs de comparaison en Visual Basic

Opérateurs de concaténation (Visual Basic)

Les opérateurs de concaténation définis dans Visual Basic sont les suivants.

&, opérateur

+, opérateur

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

System.Text

StringBuilder

Concepts

Opérateurs de concaténation (Visual Basic)

Opérateurs logiques/de bits

Les opérateurs de bits/logiques définis dans Visual Basic sont les suivants.

And, opérateur (Visual Basic)

Not, opérateur (Visual Basic)

Or, opérateur (Visual Basic)

Xor, opérateur (Visual Basic)

AndAlso, opérateur

OrElse, opérateur

IsFalse, opérateur

IsTrue, opérateur

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

Concepts

Opérateurs de bits et opérateurs logiques en Visual Basic

Opérateurs divers

Voici les divers opérateurs définis en Visual Basic.

AddressOf, opérateur

GetType, opérateur

TypeOf, opérateur (Visual Basic)

Voir aussi

Référence

Opérateurs répertoriés par fonctionnalité

Propriétés (Visual Basic)

Cette page répertorie les propriétés qui sont membres de modules Visual Basic. D'autres propriétés membres d'objets Visual Basic spécifiques sont répertoriées ailleurs dans la table des matières Microsoft Help. Pour consulter ces propriétés, accédez à la page Objets (Visual Basic) et synchronisez-la à la table des matières. Ouvrez le nœud "Objets" dans le contenu pour afficher une liste d'objets Visual Basic. Les propriétés sont répertoriées sous leurs objets respectifs.

Dans cette section

DateString, propriété

Now, propriété

ScriptEngine, propriété

ScriptEngineBuildVersion, propriété

ScriptEngineMajorVersion, propriété

ScriptEngineMinorVersion, propriété

TimeOfDay, propriété

Timer, propriété

TimeString, propriété

Today, propriété

Rubriques connexes

Référence Visual Basic

Visual Basic

DateString, propriété

Retourne ou définit une valeur **String** représentant la date actuelle indiquée par votre système.

```
Public Property DateString As String
```

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	5	Format non valide utilisé pour définir la valeur de **DateString**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs (vous pouvez comparer le numéro d'erreur et la propriété Number (objet Err)). Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par
Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

DateString retourne toujours la date système sous la forme "MM-dd-yyyy", avec une abréviation du nom du mois. Les formats acceptés pour définir la date sont "M-d-yyyy", "M-d-y", "M/d/yyyy" et "M/d/y". Ces formats sont de culture dite indifférente, ce qui signifie qu'ils ne changent pas même si vous modifiez les **Options régionales** dans le **Panneau de configuration**.

Pour obtenir ou définir l'heure système actuelle sous forme de **String**, utilisez la propriété TimeString.

Pour obtenir la date ou l'heure système actuelle au format de vos paramètres régionaux ou dans un format personnalisé, fournissez la propriété Now à la fonction Format, en spécifiant les formats de date/heure prédéfinis (fonction Format) ou les formats de date/heure définis par l'utilisateur (fonction Format). C'est ce qu'illustre l'exemple suivant.

```
MsgBox("The formatted date is " & Format(Now, "dddd, d MMM yyyy"))
```

Pour accéder à la date système actuelle sous forme de **Date**, utilisez la propriété Today.

Remarque de sécurité
La définition de la date ou de l'heure système nécessite une autorisation de code non managée, ce qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

L'exemple suivant utilise la propriété **DateString** pour afficher la date système actuelle.

VB

```
MsgBox("The current date is " & DateString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Dans la mesure où **DateString** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **DateString**.

Notes du développeur sur Smart Device

Vous pouvez obtenir, mais pas définir, la date système à l'aide de la propriété **DateString**.

Voir aussi

Référence

Now, propriété
TimeString, propriété
Today, propriété
Type de données date (Visual Basic)
DateTime

Now, propriété

Retourne une valeur **Date** contenant la date et l'heure en cours indiquées par votre système.

```
ReadOnly Public Property Now() As DateTime
```

Notes

Pour définir la date système, utilisez la propriété Today. Pour définir l'heure système, utilisez la propriété TimeOfDay.

Exemple

L'exemple suivant utilise la propriété **Now** pour retourner la date et l'heure actuelles du système.

VB

```
Dim ThisMoment As Date
' The following statement calls the Get procedure of the Visual Basic Now property.
ThisMoment = Now
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Du fait que **Now** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **Now**.

Voir aussi

Référence

Day, fonction (Visual Basic)
Hour, fonction (Visual Basic)
Minute, fonction
Month, fonction (Visual Basic)
Second, fonction (Visual Basic)
Weekday, fonction (Visual Basic)
Year, fonction (Visual Basic)
Type de données date (Visual Basic)
DateTime

ScriptEngine, propriété

Retourne une **String** représentant le runtime en cours d'utilisation.

```
ReadOnly Public Property ScriptEngine As String
```

Notes

La propriété **ScriptEngine** retourne la chaîne "VB".

Vous pouvez utiliser **ScriptEngine** lorsque votre application s'exécute comme un programme autonome, dans un script ou dans une application hôte.

Exemple

L'exemple ci-dessous utilise la propriété **ScriptEngine** et les propriétés associées pour retourner une chaîne décrivant les informations relatives au runtime utilisé :

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

Notes du développeur sur Smart Device

Cette propriété n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Globals

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Du fait que **ScriptEngine** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **ScriptEngine**.

Voir aussi

Référence

ScriptEngineBuildVersion, propriété
ScriptEngineMajorVersion, propriété
ScriptEngineMinorVersion, propriété

ScriptEngineBuildVersion, propriété

Retourne un **Integer** contenant le numéro de build du runtime en cours d'utilisation.

```
ReadOnly Public Property ScriptEngineBuildVersion As Integer
```

Notes

La valeur retournée correspond directement aux informations de version contenues dans la DLL du runtime utilisé.

Vous pouvez utiliser **ScriptEngineBuildVersion** lorsque votre application s'exécute comme un programme autonome, dans un script ou dans une application hôte.

Exemple

L'exemple ci-dessous utilise la propriété **ScriptEngineBuildVersion** et les propriétés associées afin de retourner une chaîne décrivant les informations relatives au runtime utilisé :

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

Notes du développeur sur Smart Device

Cette propriété n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Globals

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Du fait que **ScriptEngineBuildVersion** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **ScriptEngineBuildVersion**.

Voir aussi

Référence

ScriptEngine, propriété
ScriptEngineMajorVersion, propriété
ScriptEngineMinorVersion, propriété

ScriptEngineMajorVersion, propriété

Retourne un **Integer** contenant le numéro de version principale du runtime en cours d'utilisation.

```
ReadOnly Public Property ScriptEngineMajorVersion As Integer
```

Notes

La valeur retournée correspond directement aux informations de version contenues dans la DLL du runtime utilisé.

Vous pouvez utiliser **ScriptEngineMajorVersion** lorsque votre application s'exécute comme un programme autonome, dans un script ou dans une application hôte.

Exemple

L'exemple ci-dessous utilise la propriété **ScriptEngineMajorVersion** et les propriétés associées pour retourner une chaîne décrivant les informations relatives au runtime utilisé :

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

Notes du développeur sur Smart Device

Cette propriété n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Globals

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Dans la mesure où **ScriptEngineMajorVersion** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **ScriptEngineMajorVersion**.

Voir aussi

Référence

ScriptEngine, propriété
ScriptEngineBuildVersion, propriété
ScriptEngineMinorVersion, propriété

ScriptEngineMinorVersion, propriété

Retourne un **Integer** contenant le numéro de version secondaire du runtime en cours d'utilisation.

```
ReadOnly Public Property ScriptEngineMinorVersion As Integer
```

Notes

La valeur retournée correspond directement aux informations de version contenues dans la DLL du runtime utilisé.

Vous pouvez utiliser **ScriptEngineMinorVersion** lorsque votre application s'exécute comme un programme autonome, dans un script ou dans une application hôte.

Exemple

L'exemple ci-dessous utilise la propriété **ScriptEngineMinorVersion** et les propriétés associées pour retourner une chaîne décrivant les informations relatives au runtime utilisé :

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

Notes du développeur sur Smart Device

Cette propriété n'est pas prise en charge.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Globals

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Étant donné que **ScriptEngineMinorVersion** est un membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **ScriptEngineMinorVersion**.

Voir aussi

Référence

ScriptEngine, propriété
ScriptEngineBuildVersion, propriété
ScriptEngineMajorVersion, propriété

TimeOfDay, propriété

Retourne ou définit une valeur **Date** contenant l'heure en cours indiquée par votre système.

```
Public Property TimeOfDay() As DateTime
```

Notes

Le type de données **Date** inclut des composants date. Lorsqu'elle retourne l'heure système, la propriété **TimeOfDay** affecte à tous ces composants la valeur 1, de sorte que la valeur retournée représente le premier jour de l'année 1. Lorsqu'elle définit l'heure système, **TimeOfDay** ignore les composants date.

Pour accéder à l'heure système actuelle sous forme de **String**, utilisez la propriété TimeString.

Pour obtenir ou définir la date système actuelle, utilisez la propriété Today.

Remarque de sécurité

La définition de la date ou de l'heure système nécessite une autorisation de code non managée, qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

Cet exemple utilise la propriété **TimeOfDay** pour retourner l'heure système en cours.

VB

```
Dim currentTime As Date
currentTime = TimeOfDay
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Dans la mesure où **TimeOfDay** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **TimeOfDay**.

Notes du développeur sur Smart Device

Vous pouvez obtenir, mais pas définir, l'heure actuelle du jour à l'aide de la propriété **TimeOfDay**.

Voir aussi

Référence

Timer, propriété
Type de données date (Visual Basic)
DateTime

Timer, propriété

Retourne une valeur de type **Double** représentant le nombre de secondes écoulées depuis minuit.

```
ReadOnly Public Property Timer() As Double
```

Notes

La propriété **Timer** retourne à la fois les secondes et les millisecondes écoulées depuis minuit. Les secondes font partie de la partie entière de la valeur de retour, les millisecondes de la partie décimale.

Exemple

Cet exemple utilise la propriété **Timer** pour suspendre l'application. D'autres traitements peuvent être effectués lors de la pause.

VB

```
Public Sub waitFiveSeconds()
    If TimeOfDay >= #11:59:55 PM# Then
        MsgBox("The current time is within 5 seconds of midnight" & _
            vbCrLf & "The timer returns to 0.0 at midnight")
        Return
    End If
    Dim start, finish, totalTime As Double
    If (MsgBox("Press Yes to pause for 5 seconds", MsgBoxStyle.YesNo)) _
        = MsgBoxResult.Yes Then
        start = Microsoft.VisualBasic.DateAndTime.Timer
        ' Set end time for 5-second duration.
        finish = start + 5.0
        Do While Microsoft.VisualBasic.DateAndTime.Timer < finish
        ' Do other processing while waiting for 5 seconds to elapse.
        Loop
        totalTime = Microsoft.VisualBasic.DateAndTime.Timer - start
        MsgBox("Paused for " & totalTime & " seconds")
    End If
End Sub
```

Notez que vous devez qualifier la propriété **Timer** avec l'espace de noms Microsoft.VisualBasic, parce que **Timer** est également une classe définie dans les espaces de noms System.Threading, System.Timers et System.Windows.Forms.

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Dans la mesure où **Timer** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **Timer**.

Voir aussi

Référence

Randomize, fonction (Visual Basic)
TimeOfDay, propriété
Today, propriété
System
System.Windows.Forms
DateTime
ArgumentException
ArgumentOutOfRangeException

TimeString, propriété

Retourne ou définit une valeur **String** représentant l'heure en cours indiquée par votre système.

```
Public Property TimeString As String
```

Exceptions

Type d'exception	Numéro de l'erreur	Condition
InvalidCastException	5	Format non valide utilisé pour définir la valeur de **TimeString**.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion non structurée des erreurs (vous pouvez comparer le numéro d'erreur et la propriété Number (objet Err)). Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreurs par
Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

La propriété **TimeString** retourne toujours l'heure système sous la forme "HH:mm:ss", qui est un format 24 heures. Ce format est de culture dite indifférente, ce qui signifie qu'il ne change pas même si vous modifiez les **Options régionales** dans le **Panneau de configuration**.

Pour obtenir ou définir la date système actuelle sous forme de **String**, utilisez la propriété DateString.

Pour obtenir la date ou l'heure système actuelle au format de vos paramètres régionaux ou dans un format personnalisé, fournissez la propriété Now à la fonction Format, en spécifiant les formats de date/heure prédéfinis (fonction Format) ou les formats de date/heure définis par l'utilisateur (fonction Format). C'est ce qu'illustre l'exemple suivant.

```
MsgBox("The formatted time is " & Format(Now, "hh.mm.ss.fff tt"))
```

Pour accéder à l'heure système actuelle sous forme de **Date**, utilisez la propriété TimeOfDay.

Remarque de sécurité

La définition de la date ou de l'heure système nécessite une autorisation de code non managée, ce qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

Cet exemple utilise la propriété **TimeString** pour afficher l'heure système en cours.

VB

```
MsgBox("The current time is " & TimeString)
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Du fait que **TimeString** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **TimeString**.

Notes du développeur sur Smart Device

Vous pouvez obtenir, mais pas définir, l'heure système à l'aide de la propriété **TimeString**.

Voir aussi
Référence

Now, propriété
DateString, propriété
TimeOfDay, propriété
Type de données date (Visual Basic)
DateTime

Today, propriété

Retourne ou définit une valeur **Date** représentant la date actuelle indiquée par votre système.

```
Public Property Today() As DateTime
```

Notes

Le type de données **Date** inclut des composants heure. Lorsqu'elle retourne la date système, **Today** affecte à tous ces composants la valeur 0, ce qui donne une valeur retournée représentant minuit (00:00:00). Lorsqu'elle définit la date système, **Today** ignore les composants heure.

Pour accéder à la date système actuelle sous forme de **String**, utilisez la propriété DateString.

Pour obtenir ou définir la date système actuelle, utilisez la propriété TimeOfDay.

Remarque de sécurité

La définition de la date ou de l'heure système nécessite une autorisation de code non managée, ce qui peut affecter son exécution dans les situations d'un niveau de confiance partiel. Pour plus d'informations, consultez SecurityPermission et Autorisations d'accès du code.

Exemple

Cet exemple utilise la propriété **Today** pour retourner la date système actuelle.

VB

```
Dim thisDate As Date
thisDate = Today
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : DateAndTime

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Du fait que **Today** est membre d'un module plutôt que d'une classe, vous n'avez pas besoin de créer un objet sur lequel accéder à **Today**.

Notes du développeur sur Smart Device

Vous pouvez obtenir, mais pas définir, la date système actuelle à l'aide de la propriété **Today**.

Voir aussi

Référence

Format, fonction
Now, propriété
Type de données date (Visual Basic)
DateTime

Instructions (Visual Basic)

Les rubriques de cette section contiennent des tableaux des déclarations et instructions exécutables Visual Basic, ainsi que des listes importantes qui s'appliquent à de nombreuses instructions.

Dans cette section

Instructions A-E

Instructions F-P

Instructions Q-Z

Contextes de déclaration et niveaux d'accès par défaut

Liste d'attributs

Liste de paramètres

Liste de types

Rubriques connexes

Référence Visual Basic

Visual Basic

Instructions A-E

Le tableau suivant énumère les instructions du langage Visual Basic.

AddHandler	Call	Class	Const
Continue	Declare	Delegate	Dim
Do...Loop	End	Enum	Erase
Error	Event	Exit	

Voir aussi

Référence

Instructions F-P

Instructions Q-Z

Autres ressources

Référence Visual Basic

AddHandler, instruction

Associe un événement à un gestionnaire d'événements au moment de l'exécution.

```
AddHandler event, AddressOf eventhandler
```

Éléments

event

Nom de l'événement à gérer.

eventhandler

Nom de la procédure qui gère l'événement.

Notes

Les instructions **AddHandler** et **RemoveHandler** vous permettent de débuter et d'arrêter la gestion d'un événement à tout moment au cours de l'exécution du programme.

La signature de la procédure *eventhandler* doit correspondre à celle de l'événement *event*.

Le mot clé **Handles** et l'instruction **AddHandler** permettent de spécifier que des procédures particulières gèrent des événements particuliers, mais il existe des différences. L'instruction **AddHandler** connecte les procédures aux événements au moment de l'exécution. Utilisez le mot clé **Handles** lors de la définition d'une procédure pour spécifier qu'il gère un événement particulier. Pour plus d'informations, consultez Handles.

Pour les événements personnalisés, l'instruction **AddHandler** appelle l'accesseur **AddHandler** de l'événement. Pour plus d'informations sur les événements personnalisés, consultez Event, instruction.

Exemple

VB

```
Sub TestEvents()
    Dim Obj As New Class1
    ' Associate an event handler with an event.
    AddHandler Obj.Ev_Event, AddressOf EventHandler
    ' Call the method to raise the event.
    Obj.CauseSomeEvent()
    ' Stop handling events.
    RemoveHandler Obj.Ev_Event, AddressOf EventHandler
    ' This event will not be handled.
    Obj.CauseSomeEvent()
End Sub

Sub EventHandler()
    ' Handle the event.
    MsgBox("EventHandler caught event.")
End Sub

Public Class Class1
    ' Declare an event.
    Public Event Ev_Event()
    Sub CauseSomeEvent()
        ' Raise an event.
        RaiseEvent Ev_Event()
    End Sub
End Class
```

Voir aussi

Référence

RemoveHandler, instruction

Handles

Event, instruction

Concepts

Événements et gestionnaires d'événements

AddHandler et RemoveHandler

Call, instruction (Visual Basic)

Transfère le contrôle à une procédure **Function**, **Sub** ou à une procédure de bibliothèque de liens dynamiques (DLL, Dynamic-Link Library).

```
[ Call ] procedureName [ (argumentList) ]
```

Éléments

procedureName

Obligatoire. Nom de la procédure à appeler.

argumentList

Facultatif. Liste de variables ou d'expressions représentant les arguments passés à la procédure quand elle est appelée. Les arguments multiples sont séparés par des virgules. Si vous incorporez *argumentList*, vous devez le placer entre parenthèses.

Notes

Vous utilisez normalement l'instruction **Call** pour appeler une procédure qui ne retourne pas de valeur. Si la procédure retourne une valeur, l'instruction **Call** l'ignore.

Vous n'êtes pas obligé d'utiliser l'instruction **Call** lorsque vous appelez une procédure. Toutefois, elle améliore la lisibilité de votre code.

Exemple

Cet exemple illustre comment l'instruction **Call** est utilisée pour passer le contrôle à une procédure **Sub**, une fonction intrinsèque et une procédure de DLL.

VB

```
        ' (1) Call a Sub procedure.
        Call printToDebugWindow("Hello World")
...
    ' The above statement passes control to the following Sub procedure.
    Sub printToDebugWindow(ByVal anyString As String)
        Debug.WriteLine(anyString)
    End Sub
```

VB

```
' (2) Call a Visual Basic run-time function (Shell), discard the return value.
Call Shell("C:\WINNT\system32\calc.exe", AppWinStyle.NormalFocus)
' The preceding path is for Windows 2000;
' The Windows XP path is C:\Windows\system32\calc.exe.
```

VB

```
' (3) Call a Microsoft Windows DLL procedure. The Declare statement
' must be Private in a class, not in a module.
Private Declare Sub MessageBeep Lib "User32" (ByVal N As Integer)
Sub callBeepDll()
    Call MessageBeep(-1)
End Sub
```

Voir aussi

Référence

Function, instruction (Visual Basic)

Sub, instruction (Visual Basic)

Declare, instruction

Class, instruction (Visual Basic)

Déclare le nom d'une classe et introduit la définition des variables, des propriétés, des événements et des procédures que la classe contient.

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable ] [ Parti
al ] _
Class name [ ( Of typelist ) ]
    [ Inherits classname ]
    [ Implements interfacenames ]
    [ statements ]
End Class
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protégé
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

Shadows

Facultatif. Consultez Shadows.

MustInherit

Facultatif. Consultez MustInherit.

NotInheritable

Facultatif. Consultez NotInheritable.

Partial

Facultatif. Indique une définition partielle de la classe. Consultez Partial (Visual Basic).

name

Obligatoire. Nom de cette classe. Consultez Noms d'éléments déclarés.

Of

Facultatif. Spécifie qu'il s'agit d'une classe générique.

typelist

Requis si vous utilisez le mot clé Of. Liste de paramètres de type pour cette classe. Consultez Liste de types.

Inherits

Facultatif. Indique que cette classe hérite des membres d'une autre classe. Consultez l'instruction Inherits.

classname

Requis si vous utilisez l'instruction **Inherits**. Nom de la classe dont cette classe dérive.

Implements

Facultatif. Indique que cette classe implémente les membres d'une ou plusieurs interfaces. Consultez l'instruction Implements.

interfacenames

Requis si vous utilisez l'instruction **Implements**. Noms des interfaces que cette classe implémente.

statements

Facultatif. Instructions qui définissent les membres de cette classe.

End Class

Obligatoire. Met fin à la définition **Class**.

Notes

Une instruction **Class** définit un nouveau type de données. Une *classe* est un bloc de construction fondamental d'une programmation orientée objet (OOP). Pour plus d'informations, consultez Programmation orientée objet dans Visual Basic et Classes : modèles d'objets.

Vous pouvez utiliser **Class** uniquement au niveau de l'espace de noms ou du module. Cela signifie que le *contexte de déclaration* pour une classe doit être un fichier source, un espace de noms, une classe, une structure, un module ou une interface, et ne peut pas être une procédure ni un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Chaque instance d'une classe a une durée de vie indépendante de toutes les autres instances. Cette durée de vie commence lorsqu'elle est créée par une clause New (Visual Basic) ou par une fonction telle que CreateObject, fonction (Visual Basic). Elle prend fin lorsque la valeur Nothing (Visual Basic) ou les instances d'autres classes ont été affectées à toutes les variables pointant sur l'instance.

Les classes prennent par défaut l'accès Friend (Visual Basic). Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Règles

- **Imbrication.** Vous pouvez définir une classe dans une autre. La classe externe est appelée *classe conteneur* et la classe interne *classe imbriquée*.
- **Héritage.** Si la classe utilise l'instruction Inherits, vous pouvez spécifier une seule classe ou interface de base. Une classe ne peut pas hériter de plusieurs éléments.

  Une classe ne peut pas hériter d'une autre classe avec un niveau d'accès plus restrictif. Par exemple, une classe **Public** ne peut pas hériter d'une classe **Friend**.

  Une classe ne peut pas hériter d'une classe imbriquée dans celle-ci.
- **Implémentation.** Si la classe utilise l'instruction Implements, vous devez implémenter chaque membre défini par chaque interface que vous spécifiez dans *interfacenames*. La nouvelle implémentation d'un membre de classe de base est une exception. Pour plus d'informations, consultez « Nouvelle implémentation » dans Implements (Visual Basic).
- **Default, propriété.** Une classe peut spécifier qu'une propriété comme étant sa *propriété par défaut*. Pour plus d'informations, consultez Propriétés par défaut.

Comportement

- **Niveau d'accès.** Dans une classe, vous pouvez déclarer chaque membre avec son propre niveau d'accès. Les membres de la classe prennent par défaut l'accès Public (Visual Basic), à l'exception des variables et des constantes qui prennent par défaut l'accès Private (Visual Basic). Lorsqu'une classe a un accès plus limité que l'un de ses membres, le niveau d'accès de la classe a la priorité.
- **Portée.** Une classe est dans la portée dans tout son espace de noms, sa classe, sa structure ou son module conteneur.

  La portée de chaque membre de classe est la classe entière.

  Durée de vie. Visual Basic ne prend pas en charge des classes statiques. L'équivalent fonctionnel d'une classe statique est fourni par un module. Pour plus d'informations, consultez Module, instruction.

  Les durées de vie des membres de classe dépendent de la manière et de l'emplacement où elles sont déclarées. Pour plus

d'informations, consultez Durée de vie dans Visual Basic.

- **Qualification.** Le code à l'extérieur d'une classe doit qualifier le nom d'un membre portant le même nom que cette classe.

  Si le code à l'intérieur d'une classe imbriquée établit une référence non qualifiée à un élément de programmation, Visual Basic recherche d'abord l'élément dans sa classe imbriquée, puis dans sa classe conteneur, etc. jusqu'à l'élément conteneur le plus éloigné. Pour plus d'informations, consultez Résolution d'une référence lorsque plusieurs variables ont le même nom.

Classes et Modules

Ces éléments présentent beaucoup de similitudes, mais ils comportent également beaucoup de différences.

- **Terminologie.** Les versions précédentes de Visual Basic reconnaissent deux types de modules : *modules de classe* (fichiers .cls) et *modules standard* (fichiers .bas). La version actuelle appelle respectivement ces *classes* et ces *modules*.
- **Membres partagés.** Vous pouvez contrôler si un membre d'une classe est un membre partagé ou un membre d'instance.
- **Orientation objet.** Les classes sont orientées objet, ce qui n'est pas le cas des modules. Vous pouvez créer une ou plusieurs instances d'une classe. Pour plus d'informations, consultez Classes et modules.

Exemple

L'exemple suivant utilise une instruction **Class** pour définir une classe et plusieurs membres.

VB

```
Class bankAccount
    Shared interestRate As Decimal
    Private accountNumber As String
    Private accountBalance As Decimal
    Public holdOnAccount As Boolean = False

    Public ReadOnly Property balance() As Decimal
        Get
            Return accountBalance
        End Get
    End Property

    Public Sub postInterest()
        accountBalance = accountBalance * (1 + interestRate)
    End Sub

    Public Sub postDeposit(ByVal amountIn As Decimal)
        accountBalance = accountBalance + amountIn
    End Sub

    Public Sub postWithdrawal(ByVal amountOut As Decimal)
        accountBalance = accountBalance - amountOut
    End Sub
End Class
```

Voir aussi

Tâches

Comment : utiliser une classe générique

Référence

Interface, instruction (Visual Basic)
Module, instruction
Property, instruction

Concepts

Structures et classes
Durée de vie d'un objet : création et destruction des objets
Types génériques en Visual Basic

Autres ressources

Présentation des classes

Const, instruction (Visual Basic)

Déclare et définit une ou plusieurs constantes.

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ]
Const constantlist
```

Éléments

attributelist

Facultatif. Liste d'attributs s'appliquant à toutes les constantes déclarées dans cette instruction. Liste d'attributs est mis entre des signes "inférieur à" et "supérieur à" ("**<**" et "**>**").

accessmodifier

Facultatif. Utilisez cet élément pour spécifier le code qui peut accéder à ces constantes. Peut être Public (Visual Basic), Protected (Visual Basic), Friend (Visual Basic), **Protected Friend** ou Private (Visual Basic).

Shadows

Facultatif. Utilisez cet élément pour redéclarer et masquer un élément de programmation dans une classe de base. Consultez Shadows.

constantlist

Obligatoire. Liste des constantes qui sont déclarées dans cette instruction.

constant `[ , ` *constant* `... ]`

Chaque *constant* emploie la syntaxe et les paramètres suivants :

constantname `[ As` *datatype* `]` `=` *initializer*

Élément	Description
constantname	Obligatoire. Nom de la constante. Consultez Noms d'éléments déclarés.
datatype	Requis si **Option Strict** a la valeur **On**. Type de données de la constante.
initializer	Obligatoire. Expression qui est évaluée au moment de la compilation et assignée à la constante.

Notes

Si votre application contient une valeur qui ne change jamais, vous pouvez définir une constante nommée et l'utiliser à la place d'une valeur littérale. Il est plus facile de se rappeler d'un nom que d'une valeur. Vous pouvez définir la constante une seule fois et l'utiliser dans plusieurs zones de votre code. Si vous devez redéfinir la valeur dans une version ultérieure, l'instruction **Const** est la seule zone que vous devez modifier.

Vous pouvez utiliser **Const** seulement au niveau du module ou de la procédure. Cela signifie que le *contexte de déclaration* pour une variable doit être une classe, une structure, un module, une procédure ou un bloc, et ne peut pas être un fichier source, un espace de noms ou une interface. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les constantes locales (dans une procédure) disposent par défaut d'un accès public, et vous ne pouvez pas leur appliquer des modificateurs d'accès. Les constantes membres de la classe et du module (en dehors d'une procédure) disposent par défaut d'un accès privé, et les constantes membres de la structure disposent par défaut d'un accès public. Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès.

Règles

- **Contexte de déclaration.** Une constante déclarée au niveau du module, à l'extérieur de toute procédure, est une *constante membre* ; elle est membre de la classe, de la structure ou du module qui la déclare.

  Une constante déclarée au niveau de la procédure est une *constante locale* ; elle est locale à la procédure ou au bloc qui la déclare.

- **Attributs.** Vous ne pouvez appliquer des attributs qu'aux constantes membres, mais pas aux constantes locales. Un attribut fournit des informations aux métadonnées de l'assembly, qui ne sont pas explicites pour le stockage temporaire tel que les constantes locales.
- **Modificateurs.** Par défaut, toutes les constantes sont **Shared**, **Static** et **ReadOnly**. Vous ne pouvez pas utiliser l'un de ces mots clés lors de la déclaration d'une constante.

  Au niveau de la procédure, vous ne pouvez pas utiliser **Shadows** ou les modificateurs d'accès pour déclarer des constantes locales.
- **Constantes multiples.** Vous pouvez déclarer plusieurs constantes dans la même instruction de déclaration, en spécifiant la partie *constantname* pour chacune d'elles. Les constantes, lorsqu'il y en a plusieurs, sont séparées par des virgules.

Règles relatives aux types de données

- **Types de données.** L'instruction **Const** peut déclarer le type de données d'une variable. Vous pouvez spécifier un type de données ou le nom d'une énumération.
- **Type par défaut.** Si vous ne spécifiez pas *datatype*, la constante prend le type de données de *initializer*. Si vous spécifiez *datatype* et *initializer*, le type de données de *initializer* doit pouvoir être converti en *datatype*. Si ni l'élément *datatype* ni l'élément *initializer* n'est présent, le type de données a la valeur **Object** par défaut.
- **Types différents.** Vous pouvez spécifier différents types de données pour les constantes en utilisant une clause **As** distincte pour chaque variable que vous déclarez. Toutefois, vous ne pouvez pas déclarer plusieurs constantes comme étant du même type en utilisant une clause **As** commune.
- **Initialisation.** Vous devez initialiser la valeur de chaque constante dans *constantlist*. Vous utilisez *initializer* pour fournir une expression à assigner à la constante. L'expression peut être une combinaison de littéraux, d'autres constantes qui sont déjà définies et des membres de l'énumération qui sont déjà définis. Vous pouvez utiliser des opérateurs arithmétiques et logiques pour combiner ces éléments.

  Vous ne pouvez pas utiliser de variables ni de fonctions dans *initializer*. Toutefois, vous pouvez utiliser des mots clés de conversion, tels que **CByte** et **CShort**. Vous pouvez également utiliser **AscW** si vous l'appelez avec une constante **String** ou un argument **Char**, car l'évaluation peut se produire au moment de la compilation.

Comportement

- **Portée.** Les constantes locales sont uniquement accessibles à partir de leur procédure ou bloc. Les constantes membres sont accessibles à partir de n'importe quelle zone de leur classe, structure ou module.
- **Qualification.** Le code à l'extérieur d'une classe, d'une structure ou d'un module doit qualifier le nom d'une constante membre avec le nom de cette classe, de cette structure ou de ce module. Le code à l'extérieur d'une procédure ou d'un bloc ne peut pas faire référence aux constantes locales de cette procédure ou de ce bloc.

Exemple

L'exemple suivant utilise l'instruction **Const** pour déclarer les constantes à utiliser à la place de valeurs littérales.

VB

```
' The following statements declare constants.
Const maximum As Long = 459
Public Const helpString As String = "HELP"
Private Const startValue As Integer = 5
```

Si vous définissez une constante avec le type de données **Object**, le compilateur Visual Basic lui affecte le type de *initializer* au lieu de **Object**. Dans l'exemple suivant, la constante `naturalLogBase` a le type d'exécution **Decimal**.

VB

```
Const naturalLogBase As Object = CDec(2.7182818284)
MsgBox("Run-time type of constant naturalLogBase is " & _
    naturalLogBase.GetType.ToString())
```

L'exemple précédent utilise la méthode ToString sur l'objet Type retourné par l'opérateur GetType, car **Type** ne peut pas être converti en **String** à l'aide de **CStr**.

Voir aussi

Référence

Enum, instruction (Visual Basic)
#Const, directive
Dim, instruction (Visual Basic)
ReDim, instruction (Visual Basic)
Fonctions de conversion de types de données
Fonctions Asc, AscW

Concepts

Conversions implicites et explicites
Constantes et énumérations intrinsèques

Autres ressources

Constantes et énumérations en Visual Basic

Continue, instruction (Visual Basic)

Transfère immédiatement le contrôle vers l'itération suivante d'une boucle.

```
Continue { Do | For | While }
```

Notes

Vous pouvez effectuer un transfert de l'intérieur d'une boucle **Do**, **For** ou **While** vers l'itération suivante de cette même boucle. Le contrôle passe immédiatement au test de condition de la boucle, qui équivaut à effectuer un transfert vers l'instruction **For** ou **While** ou vers l'instruction **Do** ou **Loop** qui contient la clause **Until** ou **While**.

Vous pouvez utiliser **Continue** à n'importe quel emplacement de la boucle qui autorise les transferts. Les règles qui autorisent le transfert de contrôle sont identiques à celles de l'instruction GoTo.

Par exemple, si une boucle est entièrement contenue dans un bloc **Try**, un bloc **Catch** ou un bloc **Finally**, vous pouvez utiliser **Continue** pour effectuer un transfert hors de la boucle. En revanche, si la structure **Try**...**End Try** est contenue dans la boucle, vous ne pouvez pas utiliser **Continue** pour transférer le contrôle hors du bloc **Finally**, et vous pouvez l'utiliser pour le transférer hors d'un bloc **Try** ou **Catch** uniquement si le transfert s'effectue entièrement hors de la structure **Try**...**End Try**

Si vous disposez de boucles imbriquées du même type, par exemple, une boucle **Do** dans une autre boucle **Do**, une instruction **Continue Do** passe à l'itération suivante de la boucle **Do** la plus profonde qui la contient. Vous ne pouvez pas utiliser **Continue** pour passer à l'itération suivante d'une boucle conteneur du même type.

Si vous disposez de boucles imbriquées de types différents, par exemple une boucle **Do** dans une boucle **For**, vous pouvez passer à l'itération suivante de l'une ou l'autre des boucles à l'aide de **Continue Do** ou **Continue For**.

Exemple

L'exemple de code suivant utilise l'instruction **Continue While** pour passer à la colonne suivante d'un tableau si un diviseur est nul. L'instruction **Continue While** est à l'intérieur d'une boucle **For**. Elle effectue un transfert vers l'instruction `While col < lastcol`, qui est l'itération suivante de la boucle **While** la plus profonde qui contient la boucle **For**.

VB

```
Dim row, col As Integer
Dim lastrow As Integer = 6
Dim lastcol As Integer = 10
Dim a(,) As Double = New Double(lastrow, lastcol) {}
Dim b(7) As Double
row = -1
While row < lastrow
    row += 1
    col = -1
    While col < lastcol
        col += 1
        a(row, col) = 0
        For i As Integer = 0 To b.GetUpperBound(0)
            If b(i) = col Then
                Continue While
            Else
                a(row, col) += (row + b(i)) / (col - b(i))
            End If
        Next i
    End While
End While
```

Voir aussi

Tâches

Comment : passer à l'itération suivante d'une boucle

Référence

Do...Loop, instruction (Visual Basic)

For...Next, instruction (Visual Basic)

While...End While, instruction (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)

Declare, instruction

Déclare une référence à une procédure implémentée dans un fichier externe.

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ Overloads ] _
Declare [ charsetmodifier ] [ Sub ] name Lib "libname" _
[ Alias "aliasname" ] [ ([ parameterlist ]) ]
' -or-
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ Overloads ] _
Declare [ charsetmodifier ] [ Function ] name Lib "libname" _
[ Alias "aliasname" ] [ ([ parameterlist ]) ] [ As returntype ]
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

Shadows

Facultatif. Reportez-vous à Shadows.

charsetmodifier

Facultatif. Spécifie le jeu de caractères et les informations de recherche de fichier. Il peut s'agir de l'une des valeurs suivantes :

- Ansi (par défaut)
- Unicode (Visual Basic)
- Auto

Sub

Facultatif, mais **Sub** ou **Function** doit être indiqué. Indique que la procédure externe ne retourne pas de valeur.

Function

Facultatif, mais **Sub** ou **Function** doit être indiqué. Indique que la procédure externe retourne une valeur.

name

Obligatoire. Nom de cette référence externe. Pour plus d'informations, consultez Noms d'éléments déclarés.

Lib

Obligatoire. Introduit une clause **Lib** qui spécifie le fichier contenant la procédure déclarée.

libname

Obligatoire. Nom du fichier contenant la procédure déclarée.

Alias

Facultatif. Indique que la procédure déclarée ne peut pas être identifiée dans son fichier par le nom spécifié dans *name*. Vous spécifiez son identification dans *aliasname*.

aliasname

Requis si vous utilisez le mot clé **Alias**. Chaîne qui identifie la procédure de l'une des deux manières suivantes :

Nom du point d'entrée de la procédure dans son fichier, entre guillemets (**""**)

- ou -

Signe dièse (**#**) suivi d'un entier qui spécifie le nombre ordinal du point d'entrée de la procédure dans son fichier

parameterlist

Requis si la procédure prend des paramètres. Consultez Liste de paramètres.

returntype

Obligatoire si **Function** est spécifié et que **Option Strict** a la valeur **On**. Type de données de la valeur retournée par la procédure.

Notes

Parfois, vous devez appeler une procédure définie dans un fichier (par exemple une DLL ou une ressource de code) en dehors de votre projet. Lorsque vous procédez ainsi, le compilateur Visual Basic n'a pas accès aux informations nécessaires pour appeler la procédure correctement, à savoir l'emplacement de la procédure, son mode d'identification, sa séquence d'appel et son type de retour, ainsi que le jeu de caractères qu'elle utilise. L'instruction **Declare** crée une référence à une procédure externe et fournit ces informations nécessaires.

Vous pouvez utiliser **Declare** seulement au niveau du module. Cela signifie que le *contexte de déclaration* pour une référence externe doit être une classe, une structure ou un module, et ne peut pas être un fichier source, un espace de noms, une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les références externes prennent par défaut l'accès Public (Visual Basic). Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès.

Règles

- **Attributs.** Vous pouvez appliquer des attributs à une référence externe. Les attributs que vous appliquez n'ont effet que sur votre projet et non sur le fichier externe.
- **Modificateurs.** Les procédures externes sont implicitement Shared (Visual Basic). Vous ne pouvez pas utiliser le mot clé **Shared** lorsque vous déclarez une référence externe et vous ne pouvez pas modifier son état partagé.

  Une procédure externe ne peut pas participer à la substitution, à l'implémentation des membres d'interface ou à la gestion d'événements. Par conséquent, vous ne pouvez pas utiliser le mot clé **Overrides**, **Overridable**, **NotOverridable**, **MustOverride**, **Implements** ou **Handles** dans une instruction **Declare**.
- **Nom de la procédure externe.** Il n'est pas nécessaire d'attribuer à cette référence externe le même nom (dans *name*) que celui du point d'entrée de la procédure dans son fichier externe (*aliasname*). Vous pouvez utiliser une clause **Alias** pour spécifier le nom du point d'entrée. Elle peut s'avérer utile si la procédure externe porte le même nom qu'un modificateur réservé Visual Basic, une variable, une procédure ou tout autre élément de programmation de la même portée.

Remarque
Dans la plupart des DLL, les noms de points d'entrée respectent les majuscules et les minuscules.

- **Numéro de la procédure externe.** Vous pouvez également utiliser une clause **Alias** pour spécifier le nombre ordinal du point d'entrée dans la table d'exportation du fichier externe. Pour ce faire, commencez *aliasname* par un signe dièse (**#**). Ce procédé peut s'avérer utile si un caractère du nom de la procédure externe n'est pas admis en Visual Basic ou si le fichier externe exporte la procédure sans nom.

Règles relatives aux types de données

- **Types de données des paramètres.** Si **Option Strict** a la valeur **On**, vous devez spécifier le type de données de chaque paramètre dans *parameterlist*. Il peut s'agir de n'importe quel type de données ou du nom d'une énumération, d'une structure, d'une classe ou d'une interface. Dans *parameterlist*, vous utilisez une clause **As** pour spécifier le type de

données de l'argument à passer à chaque paramètre.

> **Remarque**
>
> Si la procédure externe n'a pas été écrite pour .NET Framework, vous devez veiller à ce que les types de données correspondent. Si, par exemple, vous déclarez une référence à une procédure Visual Basic 6.0 avec un argument **Integer** (16 bits dans Visual Basic 6.0), vous devez identifier l'argument correspondant comme **Short** dans l'instruction **Declare**, puisqu'il s'agit du type entier de 16 bits dans Visual Basic. De même, **Long** a une largeur de données différente dans Visual Basic 6.0 et **Date** est implémenté différemment.

- **Type de données retournées.** Si la procédure externe est un **Function** et que **Option Strict** a la valeur **On**, vous devez spécifier le type de données de la valeur retournée au code appelant. Il peut s'agir de n'importe quel type de données ou du nom d'une énumération, d'une structure, d'une classe ou d'une interface.

> **Remarque**
>
> Le compilateur Visual Basic ne vérifie pas que vos types de données sont compatibles avec ceux de la procédure externe. En cas d'incompatibilité, le Common Language Runtime génère une exception MarshalDirectiveException au moment de l'exécution.

- **Types de données par défaut.** Si **Option Strict** a la valeur **Off** et que vous ne spécifiez pas le type de données d'un paramètre dans *parameterlist*, le compilateur Visual Basic convertit l'argument correspondant vers le type de données Object. De même, si vous ne spécifiez pas *returntype*, le compilateur considère le type de données retournées comme **Object**.

> **Remarque**
>
> Étant donné que vous gérez une procédure externe qui a pu être écrite sur une plate-forme différente, il est risqué de faire des hypothèses sur les types de données ou de leur affecter des valeurs par défaut. Il est beaucoup plus sûr de spécifier le type de données de chaque paramètre et de la valeur de retour, le cas échéant. La lisibilité de votre code est également améliorée.

Comportement

- **Portée.** Une référence externe est à portée dans toute sa classe, sa structure ou son module.
- **Durée de vie.** Une référence externe a la même durée de vie que la classe, la structure ou le module dans lequel elle est déclarée.
- **Appel d'une procédure externe.** Vous appelez une procédure externe de la même manière que vous appelez une procédure **Function** ou **Sub**, en l'utilisant dans une expression si elle retourne une valeur ou en la spécifiant dans un Call, instruction (Visual Basic) si elle ne retourne pas de valeur.

  Vous passez des arguments à la procédure externe exactement comme spécifié par *parameterlist* dans l'instruction **Declare**. Ne prenez pas en compte comment les paramètres ont été initialement déclarés dans le fichier externe. De même, s'il y a une valeur de retour, utilisez-la exactement comme spécifié par *returntype* dans l'instruction **Declare**.
- **Jeux de caractères.** Vous pouvez spécifier dans *charsetmodifier* comment Visual Basic doit marshaler les chaînes lorsqu'il appelle la procédure externe. Le modificateur **Ansi** demande à Visual Basic de marshaler toutes les chaînes en valeurs ANSI, tandis que le modificateur **Unicode** lui demande de les marshaler en valeurs Unicode. Le modificateur **Auto** demande à Visual Basic de marshaler les chaînes en fonction des règles .NET Framework basées sur le *name* de la référence externe ou de *aliasname* si spécifié. La valeur par défaut est **Ansi**.

  charsetmodifier spécifie également comment Visual Basic doit rechercher la procédure externe dans son fichier externe. **Ansi** et **Unicode** demandent à Visual Basic de la rechercher sans modifier son nom au cours de la recherche. **Auto** demande à Visual Basic de déterminer le jeu de caractères de base de la plate-forme runtime et de modifier éventuellement le nom de la procédure externe comme suit :

  - sur une plate-forme ANSI, telle que Windows 95, Windows 98 ou Windows Millennium, recherchez d'abord la procédure externe sans modifier son nom. En cas d'échec, ajoutez « A » à la fin du nom de la procédure externe et recherchez-la à nouveau ;
  - sur une plate-forme Unicode, telle que Windows NT, Windows 2000 ou Windows XP, recherchez d'abord la

procédure externe sans modifier son nom. En cas d'échec, ajoutez « W » à la fin du nom de la procédure externe et recherchez-la à nouveau.

- **Mécanisme.** Visual Basic utilise le mécanisme d'*appel de plate-forme* (PInvoke) .NET Framework pour résoudre et accéder aux procédures externes. L'instruction **Declare** et la classe DllImportAttribute utilisent ce mécanisme automatiquement et il n'est pas nécessaire d'avoir des connaissances sur PInvoke. Pour plus d'informations, consultez Procédure pas à pas : appel des API Windows.

Remarque de sécurité
Si la procédure externe s'exécute en dehors du Common Language Runtime (CLR), il s'agit de *code non managé*. Lorsque vous appelez une telle procédure, par exemple une fonction API Win32 ou une méthode COM, vous pouvez exposer votre application à des risques de violation de sa sécurité. Pour plus d'informations, consultez Code non managé.

Exemple

L'exemple suivant déclare une référence externe à une procédure **Function** qui retourne le nom d'utilisateur actuel. Elle appelle ensuite la procédure externe `GetUserNameA` dans le cadre de la procédure `getUser`.

VB

```
Declare Function getUserName Lib "advapi32.dll" Alias "GetUserNameA" _
    (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Sub getUser()
    Dim buffer As String = New String(CChar(" "), 25)
    Dim retVal As Integer = getUserName(buffer, 25)
    Dim userName As String = Strings.Left(buffer, InStr(buffer, Chr(0)) - 1)
    MsgBox(userName)
End Sub
```

L'**DllImportAttribute** fournit une solution de remplacement pour l'utilisation des fonctions dans du code non managé. L'exemple suivant déclare une fonction importée sans le recours à une instruction **Declare**.

VB

```
' Add an Imports statement at the top of the class, structure, or
' module that uses the DllImport attribute.
Imports System.Runtime.InteropServices
```

VB

```
<DllImportAttribute("kernel32.dll", EntryPoint:="MoveFileW", _
    SetLastError:=True, CharSet:=CharSet.Unicode, _
    ExactSpelling:=True, _
    CallingConvention:=CallingConvention.StdCall)> _
Public Shared Function moveFile(ByVal src As String, _
  ByVal dst As String) As Boolean
    ' This function copies a file from the path src to the path dst.
    ' Leave this function empty. The DLLImport attribute forces calls
    ' to moveFile to be forwarded to MoveFileW in KERNEL32.DLL.
End Function
```

Voir aussi

Tâches

Procédure pas à pas : appel des API Windows

Référence

Imports, instruction
AddressOf, opérateur
Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Liste de paramètres
Call, instruction (Visual Basic)
LastDllError, propriété (objet Err)

Concepts

Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Type de données Integer pour les utilisateurs de Visual Basic 6.0

Delegate, instruction

Permet de déclarer un délégué. Un délégué est un type référence qui fait référence à une méthode **Shared** d'un type ou à une méthode d'instance d'un objet. Toute procédure présentant des types de paramètres et un type de retour assortis peut être utilisée pour créer une instance de cette classe déléguée. La procédure peut ensuite être appelée ultérieurement par l'intermédiaire de l'instance déléguée.

```
[ <attrlist> ] [ accessmodifier ] _
[ Shadows ] Delegate [ Sub | Function ] name [( Of typeparamlist )] [([ parameterlist ])] [
As type ]
```

Éléments

attrlist

Facultatif. Liste des attributs s'appliquant à ce délégué. Les attributs multiples sont séparés par des virgules. Vous devez mettre la Liste d'attributs entre des signes "inférieur à" et "supérieur à" ("**<**" et "**>**").

accessmodifier

Facultatif. Spécifie le code pouvant accéder à ce délégué. Il peut s'agir de l'une des valeurs suivantes :

- Public. Tout code pouvant accéder à l'élément qui déclare le délégué peut y accéder.
- Protégé. Seul le code dans la classe du délégué ou dans une classe dérivée peut y accéder.
- Fonction friend. Seul le code dans le même assembly peut accéder au délégué.
- Privé Seul le code dans l'élément qui déclare le délégué peut y accéder.

Vous pouvez spécifier **Protected Friend** pour autoriser l'accès à partir du code dans la classe du délégué, dans une classe dérivée ou dans le même assembly.

Shadows

Facultatif. Indique si ce délégué redéclare et masque un élément de programmation de même nom ou un ensemble d'éléments surchargés, dans une classe de base. Vous pouvez occulter tout type d'élément déclaré par un autre type.

Un élément occulté n'est pas disponible à partir de la classe dérivée qui l'occulte, sauf à partir de l'emplacement où l'élément d'occultation est inaccessible. Par exemple, si un élément **Private** occulte un élément de classe de base, le code qui n'est pas autorisé à accéder à l'élément **Private** accède à la place à l'élément de classe de base.

Sub

Facultatif, mais **Sub** ou **Function** doit être indiqué. Déclare cette procédure en tant que procédure **Sub** de delegate qui ne retourne pas de valeur.

Function

Facultatif, mais **Sub** ou **Function** doit être indiqué. Déclare cette procédure en tant que procédure **Function** de delegate qui retourne une valeur.

name

Requis. Nom du type délégué. Ce nom respecte les conventions standard d'affectation de noms aux variables.

typeparamlist

Facultatif. Liste de paramètres de type pour ce délégué. Les paramètres de type, lorsqu'il y en a plusieurs, sont séparés par des virgules. Vous devez mettre la Liste de types entre parenthèses et l'introduire avec le mot clé **Of**.

parameterlist

Facultatif. Liste des paramètres qui sont passés à la procédure lorsqu'elle est appelée. Vous devez mettre la Liste de paramètres entre parenthèses.

type

Requis si vous spécifiez une procédure **Function**. Type de données de la valeur de retour.

Notes

L'instruction **Delegate** définit les types de paramètres et le type de retour d'une classe déléguée. Les procédures présentant des types de paramètres et un type de retour assortis peuvent être utilisées pour créer une instance de cette classe déléguée. La procédure peut ensuite être appelée ultérieurement par l'intermédiaire de l'instance déléguée, en appelant la méthode **Invoke** du délégué.

Les délégués peuvent être déclarés au niveau de l'espace de noms, du module, de la classe ou de la structure, mais pas dans une procédure.

Chaque classe déléguée définit un constructeur auquel les spécifications d'une méthode objet sont passées. Un argument à un constructeur délégué doit être une expression au format :

AddressOf [*expression*.]*methodname*

Le type de l'*expression* au moment de la compilation doit correspondre au nom d'une classe ou d'une interface qui contient une méthode avec le nom spécifié dont la signature correspond à celle de la classe déléguée. La méthode *methodname* peut être soit une méthode partagée, soit une méthode d'instance. *methodname* n'est pas facultatif, même si vous créez un délégué pour la méthode par défaut de la classe.

Exemple

L'exemple suivant utilise l'instruction **Delegate** pour déclarer un délégué afin de fonctionner sur deux nombres et de retourner un nombre. La méthode `DelegateTest` accepte une instance d'un délégué de ce type et s'en sert pour fonctionner sur deux nombres.

VB

```
Delegate Function MathOperator( _
    ByVal x As Double, _
    ByVal y As Double _
) As Double

Function AddNumbers( _
    ByVal x As Double, _
    ByVal y As Double _
) As Double
    Return x + y
End Function

Function SubtractNumbers( _
    ByVal x As Double, _
    ByVal y As Double _
) As Double
    Return x - y
End Function

Sub DelegateTest( _
    ByVal x As Double, _
    ByVal op As MathOperator, _
    ByVal y As Double _
)
    Dim ret As Double
    ret = op.Invoke(x, y) ' Call the method.
    MsgBox(ret)
End Sub

Protected Sub Test()
    DelegateTest(5, AddressOf AddNumbers, 3)
    DelegateTest(9, AddressOf SubtractNumbers, 3)
End Sub
```

Voir aussi

Tâches

Comment : utiliser une classe générique

Référence

AddressOf, opérateur

Of

Concepts

Types génériques en Visual Basic

Autres ressources

Délégués en Visual Basic

Dim, instruction (Visual Basic)

Déclare et alloue un espace de stockage pour une ou plusieurs variables.

```
[ <attributelist> ] [ accessmodifier ] [[ Shared ] [ Shadows ] | [ Static ]] [ ReadOnly ]
Dim [ WithEvents ] variablelist
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

Shared

Facultatif. Consultez Shared.

Shadows

Facultatif. Consultez Shadows.

Static

Facultatif. Consultez Static.

ReadOnly

Facultatif. Reportez-vous à ReadOnly.

WithEvents

Facultatif. Spécifie qu'il s'agit de variables objets qui font référence aux instances d'une classe qui peut déclencher des événements. Reportez-vous à WithEvents.

variablelist

Obligatoire. Liste des variables qui sont déclarées dans cette instruction.

```
variable [ , variable ... ]
```

Chaque *variable* emploie la syntaxe et les éléments suivants :

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

Élément	Description
variablename	Obligatoire. Nom de la variable. Consultez Noms d'éléments déclarés.
boundslist	Facultatif. Liste des limites de chaque dimension d'une variable de tableau.
New	Facultatif. Crée une instance de la classe lorsque l'instruction **Dim** s'exécute.
datatype	Requis si **Option Strict** a la valeur **On**. Type de données de la variable.

initializer	Facultatif si **New** n'est pas spécifié. Expression évaluée et assignée à la variable lors de sa création.

Notes

Vous devez déclarer chaque variable que vous utilisez dans votre programme pour spécifier au compilateur Visual Basic le type de données de la variable et d'autres informations telles que le code qui peut y accéder. L'exemple suivant déclare une variable pour qu'elle contienne une valeur **Integer**.

```
Dim numberOfStudents As Integer
```

Vous pouvez utiliser **Dim** uniquement au niveau du module ou de la procédure. Cela signifie que le *contexte de déclaration* pour une variable doit être une classe, une structure, un module, une procédure ou un bloc et ne peut pas être un fichier source, un espace de noms ou une interface. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Si **Option Explicit** est **On** (la valeur par défaut), le compilateur requiert une déclaration pour chaque variable que vous utilisez. Si vous activez **Option Explicit Off**, chaque variable non déclarée prend la valeur par défaut du type de données Object, ce qui peut ne pas correspondre à ce que voulez faire.

Vous pouvez spécifier le type de données de chaque variable dans l'instruction **Dim**. Vous pouvez également spécifier une valeur initiale. Sinon, Visual Basic utilise les paramètres par défaut. Pour plus d'informations, consultez « Règles relatives aux types de données » et « Valeurs par défaut » sous « Informations détaillées » dans cette page d'aide. L'exemple suivant déclare et initialise une variable **String**.

```
Dim summary As String = "Summary of results"
```

Vous pouvez spécifier le code qui peut accéder à une variable en fournissant un *accessmodifier* dans l'instruction **Dim**. Pour plus d'informations, consultez des « Modificateurs » et« Niveau d'accès » sous « Informations détaillées » dans cette page d'aide.

Vous pouvez déclarer une variable pour qu'elle contienne un *tableau*, lequel peut contenir plusieurs valeurs. Pour plus d'informations, consultez « Règles de tableau » sous « Informations détaillées » dans cette page d'aide. Pour plus d'informations sur les tableaux, consultez Tableaux dans Visual Basic. L'exemple suivant déclare une variable de tableau **Integer**.

```
Dim days() As Integer
```

En général, vous devez placer toutes vos instructions **Dim** au début de la région de code dans laquelle vous utilisez les variables. Pour plus d'informations, consultez « Dépannage » sous « Informations détaillées » dans cette page d'aide.

Informations détaillées

Cette section aborde en détail les zones suivantes :

- Déclaration
- Types de données
- Tableaux
- Comportement
- Dépannage

Règles de déclaration

- **Contexte de déclaration.** Une variable déclarée au niveau du module, à l'extérieur de toute procédure, est une *variable membre* ou *champ* ; elle est membre de la classe, de la structure ou du module qui la déclare.

  Une variable déclarée au niveau de la procédure est une *variable locale* ; elle est locale par rapport à la procédure ou au bloc qui la déclare.

- **Attributs.** Vous ne pouvez appliquer des attributs qu'aux variables membres et non aux variables locales. Un attribut fournit des informations aux métadonnées de l'assembly, ce qui n'a pas de sens pour le stockage temporaire tel que les

variables locales.

- **Utilisation implicite de Dim.** Si vous spécifiez l'un des modificateurs **Public**, **Protected**, **Friend**, **Protected Friend**, **Private**, **Shared**, **Shadows**, **Static**, **ReadOnly** ou **WithEvents**, vous pouvez éventuellement omettre le mot clé **Dim**.

```
Public maximumAllowed As Double
Protected Friend currentUserName As String
Private salary As Decimal
Static runningTotal As Integer
```

  Au niveau du module, vous ne pouvez pas utiliser le modificateur **Static** pour déclarer des variables membres. Au niveau de la procédure, vous ne pouvez pas utiliser **Shared**, **Shadows**, **ReadOnly**, **WithEvents** ni aucun autre modificateur d'accès pour déclarer des variables locales.

- **Variables WithEvents.** Vous pouvez spécifier **WithEvents** uniquement sur les variables membres et non sur les variables locales à l'intérieur d'une procédure.

  Si vous spécifiez **WithEvents**, le type de données des variables doit être un type de classe spécifique et non **Object**. Vous ne pouvez pas déclarer un tableau avec **WithEvents**.

  Pour plus d'informations sur les événements, consultez Événements et gestionnaires d'événements.

- **Variables multiples.** Vous pouvez déclarer de nombreuses variables dans la même instruction de déclaration, en spécifiant l'élément *variablename* pour chacune d'elle et en faisant suivre chaque nom de tableau de parenthèses. Les variables multiples sont séparées par des virgules.

```
Dim lastTime, nextTime, allTimes() As Date
```

Règles relatives aux types de données

- **Types de données.** L'instruction **Dim** peut déclarer le type de données d'une variable. Vous pouvez spécifier n'importe quel type de données ou le nom d'une énumération, d'une structure, d'une classe ou d'une interface.

```
Dim finished As Boolean
Dim monitorBox As System.Windows.Forms.Form
```

- **Type par défaut.** Si vous ne spécifiez pas *datatype*, la variable prend le type de données de *initializer*. Si ni l'élément *datatype* ni l'élément *initializer* n'est présent, le type de données prend la valeur du Object, type de données par défaut. Si vous spécifiez *datatype* et *initializer*, le type de données de *initializer* doit pouvoir être converti en *datatype*.

- **Types divers.** Vous pouvez spécifier différents types de données pour différentes variables à l'aide d'une clause **As** séparée pour chaque variable que vous déclarez. Vous pouvez également déclarer plusieurs variables pour qu'elles soient du même type à l'aide d'une clause **As** commune. Chaque variable prend le type de données spécifié dans la première clause **As** rencontrée après son élément *variablename*.

```
Dim a, b, c As Single, x, y As Double, i As Integer
' a, b, and c are all Single; x and y are both Double
```

- **Initialisation.** L'instruction **Dim** peut initialiser le contenu de variables sélectionnées dans *variablelist*. Pour un type valeur, vous utilisez *initializer* pour fournir une expression à assigner à la variable. L'expression doit avoir pour valeur une constante qui peut être calculée à la compilation.

```
Dim quantity As Integer = 10
Dim message As String = "Just started"
```

  Pour un type référence, vous utilisez le mot clé **New** pour créer une instance de la classe ou structure spécifiée par *datatype*. Si vous utilisez **New**, vous n'utilisez pas d'expression *initializer*. Vous fournissez plutôt des arguments, si nécessaire, au constructeur de la classe dans laquelle vous créez la variable.

```
Dim bottomLabel As New System.Windows.Forms.Label
```

Si vous déclarez plusieurs variables avec la même clause **As**, vous ne pouvez pas spécifier un *initializer* pour ce groupe de variables.

Règles de tableau

- **Variables de tableau.** Pour spécifier qu'une variable de *variablelist* est un tableau, vous faites suivre immédiatement son *variablename* de parenthèses. Si le tableau a plusieurs dimensions, vous devez inclure des virgules entre les parenthèses pour indiquer le nombre de dimensions.

  ```
  Dim oneDimension(), twoDimensions(,), threeDimensions(,,) As Byte
  ```

  Un tableau peut contenir entre 1et 32 dimensions.

  Pour plus d'informations, consultez Tableaux dans Visual Basic.

- **Limites de tableau.** Vous pouvez spécifier la limite inférieure et supérieure de chaque dimension. Pour ce faire, vous incluez un *boundslist* à l'intérieur des parenthèses. Pour chaque dimension, le *boundslist* spécifie la limite supérieure et éventuellement la limite inférieure. La limite inférieure est toujours nulle, que vous la spécifiiez ou non. Chaque index peut varier de zéro à sa valeur de limite supérieure.

  Les deux instructions suivantes sont équivalentes. Chaque instruction déclare un tableau de 21 éléments **Integer**. Lorsque vous accédez au tableau, l'index peut varier de 0 à 20.

  ```
  Dim totals(20) As Integer
  Dim totals(0 To 20) As Integer
  ```

  L'instruction suivante déclare un tableau à deux dimensions de type **Double**. Le tableau a 4 lignes (3 + 1) de 6 colonnes (5 + 1) chacune.

  ```
  Dim matrix2(3, 5) As Double
  ```

  Notez qu'une limite supérieure représente la valeur la plus élevée possible pour cet index et non la longueur de cette dimension qui est la limite supérieure plus un.

- **Limites de tableaux vides.** Vous pouvez laisser toutes les limites vides dans une déclaration de tableau. Si vous le faites, le tableau a le nombre de dimensions que vous spécifiez, mais il n'est pas initialisé. Par conséquent, il a une valeur de **Nothing** jusqu'à ce que vous initialisiez au moins certains de ses éléments. L'instruction **Dim** doit spécifier des limites pour toutes les dimensions ou pour aucune d'entre elles.

  ```
  Dim messages() As String
  ```

- **Tableaux vides.** Il est possible d'utiliser la valeur -1 pour déclarer la limite supérieure d'une dimension d'un tableau. En d'autres termes, le tableau est vide mais sa valeur n'est pas **Nothing**. Cette distinction est requise par certaines fonctions du Common Language Runtime. Toutefois, le code Visual Basic ne peut pas accéder à un tel tableau. Si vous tentez d'y accéder, une erreur IndexOutOfRangeException se produit lors de l'exécution. Pour plus d'informations, consultez Comment : créer un tableau sans éléments.

- **Initialisation des tableaux.** Vous pouvez initialiser les valeurs d'un tableau en mettant les valeurs d'initialisation entre accolades (**{}**).

  ```
  Dim longArray() As Long = {0, 1, 2, 3}
  ```

  Pour les tableaux multidimensionnels, l'initialisation de chaque dimension séparée est entourée d'accolades dans la dimension externe. Les éléments sont spécifiés dans l'ordre ligne-champ.

  ```
  Dim twoDimensions(,) As Integer = {{0, 1, 2}, {10, 11, 12}}
  ```

Comportement

- **Valeurs par défaut.** Si vous ne spécifiez pas *initializer* pour une variable, Visual Basic initialise celle-ci à la valeur par défaut pour ce type de données. Le tableau suivant indique les valeurs d'initialisation par défaut.

Type de données	Valeur par défaut
Tous les types numériques (y compris **Byte** et **SByte**)	0
Char	Binaire 0
Tous les types référence (y compris **Object**, **String** et tous les t ableaux)	**Nothing**
Boolean	**False**
Date	1er janvier (0 heure) de l'an 1 pour (01/01/0001 12:00:0 0 AM).

  Chaque élément d'une structure est initialisé comme s'il s'agissait d'une variable séparée. Si vous déclarez la longueur d'un tableau sans initialiser ses éléments, chaque élément est initialisé comme s'il s'agissait d'une variable distincte.

- **Niveau d'accès.** Variables locales (dans une procédure) à accès public par défaut sur lesquelles vous ne pouvez pas utiliser de modificateurs d'accès.

  Les variables membres de classe ou de module (en dehors de toute procédure) prennent par défaut l'accès privé, tandis que les variables membres de structure prennent par défaut l'accès public. Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès.

- **Portée.** Les variables locales sont dans la portée de leur procédure ou bloc uniquement. Les variables membres sont dans la portée dans l'ensemble de leur classe, structure ou module.

- **Qualification.** Le code à l'extérieur d'une classe, d'une structure ou d'un module doit qualifier le nom d'une variable de membre avec le nom de cette classe, de cette structure ou de ce module. Le code en dehors d'une procédure ou d'un bloc ne peut pas faire référence à des variables locales de cette procédure ou bloc.

- **Durée de vie.** Une variable locale **Static** a une durée de vie plus longue que celle de la procédure dans laquelle elle est déclarée. Les limites de la durée de vie de la variable dépendent de l'endroit où est déclarée la procédure et si celle-ci est de type **Shared**.

Déclaration de procé dure	Variable initialisée	La variable cesse d'exister
Dans un module	Au premier appel de la procédure	Lorsque votre programme termine l'exécution
Dans une classe ou stru cture, **Shared**	Au premier appel de la procédure sur une instance spécifiqu e ou sur la classe ou la structure elle-même	Lorsque votre programme termine l'exécution
Dans une classe ou stru cture, non **Shared**	Au premier appel de la procédure sur une instance spécifiqu e	Lorsque l'instance est libérée pour le garbage collection (GC)

Dépannage

- **Ordre d'exécution.** L'instruction **Dim** n'est pas elle-même une instruction exécutable. Toutefois, si elle initialise une ou plusieurs variables, les initialisations sont traitées comme des instructions d'assignation. Cela signifie que la valeur d'une variable passe par les étapes suivantes :

  1. Sur la première entrée dans l'élément de code déclarant la variable, Visual Basic l'initialise à la valeur par défaut pour son type de données.
  2. Lorsque l'exécution arrive à l'instruction **Dim** de la variable, Visual Basic l'initialise à la valeur fournie dans

l'instruction **Dim**.

3. Chaque fois que l'exécution retourne à l'instruction **Dim** de la variable, Visual Basic lui affecte une nouvelle fois la valeur fournie dans l'instruction **Dim**.

Les conséquences sont les suivantes lorsque vous utilisez une instruction **Dim** pour initialiser une variable :

- si vous utilisez la variable avant d'exécuter l'instruction **Dim**, sa valeur est la valeur par défaut de son type de données et non la valeur fournie dans l'instruction **Dim** ;
- si l'exécution n'atteint jamais l'instruction **Dim**, la variable n'est jamais initialisée à la valeur fournie dans l'instruction **Dim** ;
- si vous modifiez la valeur et que vous retournez en revanche à l'instruction **Dim**, votre valeur modifiée est remplacée par la valeur fournie dans l'instruction **Dim** ;

si vous placez toutes vos instructions **Dim** au début de la région de code dans laquelle elles apparaissent, par exemple un module ou une procédure, votre code les initialise comme prévu.

Exemple

L'exemple suivant déclare les variables à l'aide de l'instruction **Dim** avec différentes options.

VB

```
' The following statement declares and initializes a Long variable.
Dim startingAmount As Long = 500
```

VB

```
' The following statement declares a variable that refers to a Button
' object, creates a new Button object, and assigns it to the variable.
Dim switchButton As New System.Windows.Forms.Button
```

VB

```
' The following statement declares a variable that can only be
' accessed by code in the same class, structure, or module.
Private homeTelephone As String = ""
```

VB

```
' The following statement declares a local variable that always retains
' its value, even after its procedure returns to the calling code.
Static totalSales As Double
```

VB

```
' The following statement declares a variable that refers to an array.
Dim highTemperature(31) As Integer
```

VB

```
' The following statement declares and initializes an array variable
' that holds 4 Boolean check values.
Dim checkValues() As Boolean = {False, False, True, False}
```

Voir aussi

Tâches

Comment : déclarer une variable tableau

Comment : déclarer une variable tableau

Référence

Const, instruction (Visual Basic)

ReDim, instruction (Visual Basic)

Option Explicit, instruction (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic

Do...Loop, instruction (Visual Basic)

Répète un bloc d'instructions tant qu'une condition **Boolean** a la valeur **True** ou jusqu'à ce que la condition devienne **True**.

```
Do { While | Until } condition
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop
-or-
Do
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop { While | Until } condition
```

Éléments

While

Obligatoire sauf si **Until** est utilisé. Répétez la boucle jusqu'à ce que *condition* ait la valeur **False**.

Until

Obligatoire sauf si **While** est utilisé. Répétez la boucle jusqu'à ce que *condition* ait la valeur **True**.

condition

Facultatif. Expression **Boolean**. Si *condition* a la valeur **Nothing**, Visual Basic la considère comme **False**.

statements

Facultatif. Une ou plusieurs instructions répétées tant que *condition* a la valeur **True**, ou jusqu'à ce qu'elle le devienne.

Exit Do

Facultatif. Transfère le contrôle en dehors de la boucle **Do**.

Loop

Obligatoire. Met fin à la définition de la boucle **Do**.

Notes

Utilisez une structure **Do...Loop** lorsque vous souhaitez répéter un ensemble d'instructions un nombre indéfini de fois, jusqu'à ce qu'une condition soit satisfaite. Si vous souhaitez répéter les instructions un nombre défini de fois, il est préférable d'utiliser For...Next Statement.

La structure **Do...Loop** offre plus de flexibilité que While...End While, instruction (Visual Basic) car elle vous permet de mettre fin à la boucle lorsque *condition* n'a plus la valeur **True** ou lorsqu'elle prend la valeur **True**. Elle vous permet également de tester *condition* au début ou à la fin de la boucle.

Règles

- **Type de condition.** La condition provient généralement d'une comparaison entre deux valeurs, mais elle peut être une expression qui évalue une valeur Boolean, type de données (Visual Basic) (**True** ou **False**). Elle peut comprendre des valeurs d'autres types de données, tels que des types numériques, qui ont été converties en type **Boolean**.
- **Test de la condition.** Vous pouvez tester *condition* une seule fois, au début ou à la fin de la boucle. Vous pouvez utiliser **While** ou **Until** pour spécifier *condition*, mais pas les deux.
- **Nombre d'itérations.** Si vous testez *condition* au début de la boucle (dans l'instruction **Do**), la boucle peut ne jamais s'exécuter, même une fois. Si vous la testez à la fin de la boucle (dans l'instruction **Loop**), la boucle s'exécute toujours, au moins une fois.
- **Imbrication de boucles.** Vous pouvez imbriquer des boucles **Do** en plaçant une boucle à l'intérieur d'une autre. Vous pouvez également imbriquer divers types de structures de contrôle l'un dans l'autre. Pour plus d'informations, consultez Structures de contrôle imbriquées.

- **Transfert en dehors de la boucle.** Exit, instruction (Visual Basic) passe immédiatement le contrôle à l'instruction qui suit l'instruction **Loop**. Vous pouvez vouloir quitter une boucle si vous décelez une condition qui la rend inutile ou impossible pour poursuivre l'itération, telle qu'une valeur erronée ou une demande d'arrêt. Vous pouvez placer un nombre quelconque d'instructions **Exit Do** n'importe où dans la boucle **Do**. **Exit Do** est généralement utilisé après l'évaluation d'une condition, par exemple dans une structure **If...Then...Else**.

Boucles infinies

Vous pouvez utiliser **Exit Do** pour tester une condition qui engendrerait une *boucle infinie*, c'est-à-dire une boucle qui pourrait s'exécuter de nombreuses fois, voire indéfiniment. Si vous détectez une telle condition, vous pouvez utiliser **Exit Do** pour quitter la boucle. Sinon, la boucle continue à s'exécuter.

Dans l'exemple suivant, la valeur assignée à `number` peut entraîner l'exécution de la boucle plus de 2 ^ 31 fois. L'instruction **If** vérifie cette condition et, si c'est le cas, quitte ensuite la boucle afin d'éviter le bouclage à l'infini.

VB

```
Sub exitDoExample()
    Dim counter As Integer = 0
    Dim number As Integer = 8
    Do Until number = 10
        If number <= 0 Then Exit Do
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

Remarque

Pour arrêter une boucle infinie, appuyez sur ÉCHAP ou sur CTRL+ATTN.

Exemple

L'exemple suivant illustre les structures **Do**...**Loop** imbriquées, ainsi que l'utilisation de **While** et **Until**, et le test au début et (instruction **Do**) et à la fin (instruction **Loop**) de la boucle.

VB

```
    Dim check As Boolean = True
    Dim counter As Integer = 0
    Do
        Do While counter < 20
            counter += 1
            If counter = 10 Then
                check = False
                Exit Do
            End If
        Loop
    Loop Until check = False
End Sub
```

Dans le précédent exemple, l'instruction **Do...Loop** interne se répète 10 fois, attribue la valeur **False** à l'indicateur, puis quitte la boucle prématurément à l'aide de l'instruction **Exit Do**. La boucle externe s'arrête immédiatement après vérification de la valeur de l'indicateur.

Voir aussi

Tâches

Comment : améliorer les performances d'une boucle

Référence

For...Next, instruction (Visual Basic)
Boolean, type de données (Visual Basic)
Exit, instruction (Visual Basic)
While...End While, instruction (Visual Basic)

Concepts

Structures de boucle
Structures de contrôle imbriquées

End, instruction

Termine immédiatement l'exécution.

```
End
```

Notes

Vous pouvez placer l'instruction **End** n'importe où dans une procédure pour forcer l'arrêt de toute l'application. **End** ferme tous les fichiers ouverts avec une instruction **Open** et efface les variables de l'application. L'application se ferme lorsqu'il n'y a plus aucun programme contenant des références à ses objets et qu'aucun code n'est exécuté.

Remarque

L'instruction **End** met immédiatement fin à l'exécution du code, sans appeler la méthode **Dispose** ou **Finalize** ou tout autre code Visual Basic. Les références d'objet contenues dans d'autres programmes sont annulées. Si une instruction **End** est prés ente dans un bloc **Try** ou **Catch**, le contrôle ne passe pas au bloc **Finally** correspondant.

L'instruction **Stop** interrompt l'exécution, mais contrairement à **End**, elle ne ferme pas les fichiers et n'efface pas les variables, sauf si elle est placée dans un fichier exécutable (.exe) compilé.

Étant donné que **End** met fin à votre application sans se préoccuper des ressources qui peuvent être ouvertes, vous devez essayer de la fermer correctement avant de l'utiliser. Par exemple, si votre application contient des formulaires ouverts, vous devez les fermer avant que le contrôle atteigne l'instruction **End**.

Vous devez utiliser **End** avec parcimonie, et uniquement lorsque vous devez effectuer un arrêt immédiat. Les méthodes d'arrêt normales d'une procédure (Return, instruction (Visual Basic) et Exit, instruction (Visual Basic)) permettent de fermer correctement la procédure, mais également le code appelant. Par exemple, une application console conserve simplement **Return** de la procédure **Main**.

Remarque de sécurité

L'instruction **End** appelle la méthode Exit de la classe Environment dans l'espace de noms System. **Exit** requiert que vous dis posiez d'une autorisation UnmanagedCode. Si vous ne possédez pas cette autorisation, une erreur SecurityException se prod uit.

Lorsqu'elle est suivie d'un mot clé supplémentaire, l'instruction End (Visual Basic) représente la fin de la définition de la procédure ou du bloc approprié. Par exemple, **End Function** met fin à la définition d'une procédure **Function**.

Exemple

L'exemple ci-dessous utilise l'instruction **End** pour mettre fin à l'exécution d'un code si l'utilisateur le demande.

VB

```
Sub Form_Load()
  Dim answer As MsgBoxResult
  answer = MsgBox("Do you want to quit now?", MsgBoxStyle.YesNo)
  If answer = MsgBoxResult.Yes Then
      MsgBox("Terminating program")
      End
  End If
End Sub
```

Notes du développeur sur Smart Device

Cette instruction n'est pas prise en charge.

Voir aussi

Tâches

Comment : demander l'autorisation d'accéder à du code non managé

Référence

Stop, instruction (Visual Basic)
End (Visual Basic)
SecurityPermissionFlag

Enum, instruction (Visual Basic)

Déclare une énumération et définit les valeurs de ses membres.

```
[ <attribute list> ] [ access modifier ]  [ Shadows ]
Enum enumeration name [ As data type ]
      member list
End Enum
```

Éléments

attribute list

Facultatif. Liste des attributs s'appliquant à cette énumération. Vous devez mettre la liste d'attributs entre des signes « inférieur à » et « supérieur à » (« **<** » et « **>** »).

access modifier

Facultatif. Spécifie le code pouvant accéder à cette énumération. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private

Vous pouvez spécifier **Protected Friend** pour autoriser l'accès à partir du code dans la classe de l'énumération, une classe dérivée ou le même assembly.

Shadows

Facultatif. Spécifie que cette énumération redéclare et masque un élément de programmation de même nom, ou un ensemble d'éléments surchargés, dans une classe de base. Vous pouvez uniquement spécifier Shadows sur l'énumération elle-même, et non sur l'un de ses membres.

enumeration name

Obligatoire. Nom de l'énumération. Pour plus d'informations sur les noms valides, consultez Noms d'éléments déclarés.

data type

Obligatoire si **Option Strict** a la valeur **On**. Type de données de l'énumération et de tous ses membres.

member list

Obligatoire. Liste des constantes membres qui sont déclarées dans cette instruction. Plusieurs membres apparaissent sur les lignes de code source individuelles.

Chaque *member* emploie la syntaxe et les éléments suivants : `[<attribute list>] member name [ = initializer ]`

Élément	Description
member name	Obligatoire. Nom de ce membre.
initializer	Facultatif. Expression qui est évaluée au moment de la compilation et assignée à ce membre.

End Enum

Met fin au bloc **Enum**.

Notes

Si vous avez un jeu de valeurs invariables qui sont liées logiquement les unes aux autres, vous pouvez les définir ensemble dans une énumération. Cela fournit des noms explicites pour l'énumération et ses membres, qui sont plus faciles à mémoriser que leurs valeurs. Vous pouvez ensuite utiliser les membres d'énumération à divers endroits de votre code. La lisibilité de votre programme s'en trouve également améliorée car toutes les valeurs connexes utilisent le même nom d'énumération.

Vous pouvez utiliser **Enum** uniquement au niveau de l'espace de noms ou du module. Cela signifie que le *contexte de déclaration* pour une énumération doit être un fichier source, un espace de noms, une classe, une structure, un module ou une interface, et ne peut pas être une procédure. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

L'instruction **Enum** peut déclarer le type de données d'une énumération. Chaque membre prend le type de données de l'énumération. Vous pouvez spécifier **Byte**, **Integer**, **Long**, **SByte**, **Short**, **UInteger**, **ULong** ou **UShort**.

Si vous ne spécifiez pas *initializer* pour un membre, Visual Basic l'initialise soit à zéro (si c'est le premier *member* dans *member list*), soit à une valeur supérieure d'une unité à la valeur du *member* qui précède immédiatement.

Les énumérations de membres d'une classe, d'une structure, d'un module et d'une interface disposent par défaut d'un accès public. Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès. Les énumérations de membres d'un espace de noms disposent par défaut d'un accès ami. Vous pouvez régler un niveau d'accès public, mais pas privé ni protégé. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

- **Contexte de déclaration.** Une énumération déclarée au niveau du module, à l'extérieur de toute procédure, est une *énumération membre* ; elle est membre de la classe, de la structure, du module ou de l'interface qui la déclare.

  Une énumération déclarée au niveau de l'espace de noms, à l'extérieur de toute classe, toute structure, tout module ou toute interface, est uniquement membre de l'espace de noms dans lequel elle apparaît.

- **Attributs.** Vous pouvez appliquer des attributs à l'ensemble d'une énumération, mais pas à ses membres individuellement. Un attribut fournit des informations aux métadonnées de l'assembly.

- **Modificateurs.** Par défaut, toutes les énumérations sont des types et leurs champs sont des constantes. Par conséquent, les mots clés **Shared**, **Static** et **ReadOnly** ne peuvent pas être utilisés lors de la déclaration d'une énumération ou de ses membres.

Règles relatives aux types de données

- **Type par défaut.** Si vous ne spécifiez pas *data type* pour l'énumération, chaque membre prend le type de données de son *initializer*. Si vous spécifiez *data type* et *initializer*, le type de données de *initializer* doit pouvoir être converti en *data type*. Si ni l'élément *data type* ni l'élément *initializer* n'est présent, le type de données a la valeur **Integer** par défaut.

- **Initialisation.** L'instruction **Enum** peut initialiser le contenu de membres sélectionnés dans *member list*. Vous utilisez *initializer* pour fournir une expression à assigner au membre.

  L'expression fournie dans chaque *initializer* peut être n'importe quelle combinaison de littéraux, d'autres constantes déjà définies et de membres d'énumération déjà définis, y compris un membre précédent de cette énumération. Vous pouvez utiliser des opérateurs arithmétiques et logiques pour combiner ces éléments.

  Vous ne pouvez pas utiliser de variables ni de fonctions dans *initializer*. Toutefois, vous pouvez utiliser des mots clés de conversions, tels que **CByte** et **CShort**. Vous pouvez également utiliser **AscW** si vous l'appelez avec une constante **String** ou un argument **Char**, car l'évaluation peut se produire au moment de la compilation.

Comportement

- **Niveau d'accès.** Tous les membres d'énumération ont un accès public, et vous ne pouvez pas utiliser de modificateurs d'accès sur ceux-ci. Toutefois, si l'énumération proprement dite possède a un niveau d'accès plus restreint, le niveau d'accès spécifié de l'énumération prend la priorité.

- **Portée.** Les énumérations de membres sont accessibles à partir de n'importe quel endroit dans leur classe, structure, module ou interface. Les énumérations de membres d'un espace de noms sont accessibles à partir de n'importe quel code dans cet espace de noms.

- **Qualification.** Le code à l'extérieur d'une classe, d'une structure ou d'un module doit qualifier le nom d'une énumération de membre avec le nom de cette classe, de cette structure ou de ce module.

- **Valeurs non valides.** Si la valeur d'un membre dépasse la plage autorisée pour le type de données sous-jacent ou si vous initialisez un membre à la valeur maximale autorisée par le type de données sous-jacent, le compilateur signale une erreur.

Les *variables d'énumération* sont des variables déclarées comme étant de type **Enum**. Déclarer une variable de cette manière vous aide à contrôler les valeurs que vous lui assignez. Toutefois, vous pouvez toujours assigner une valeur qui n'est pas un membre d'énumération, à condition que son type de données puisse être converti en type de données de l'énumération. Cela est utile lorsque l'énumération est un champ Indicateur et que vous assignez une combinaison d'indicateurs à la variable

d'énumération. L'exemple suivant montre une assignation de plusieurs indicateurs à une variable d'énumération.

VB

```
Enum filePermissions
    create = 1
    read = 2
    write = 4
    delete = 8
End Enum
```

VB

```
Dim file1Perm As filePermissions
file1Perm = filePermissions.create Or filePermissions.read
```

Vous devez qualifier chaque référence à un membre d'énumération, soit avec le nom d'une variable d'énumération, soit avec le nom de l'énumération. Dans le cadre de l'exemple précédent, vous pouvez faire référence au premier membre en tant que `filePermissions.create`, pas comme `create`.

Exemple

L'exemple suivant utilise l'instruction **Enum** pour définir un jeu connexe de valeurs de constantes nommées. Dans ce cas, les valeurs sont des couleurs que vous pouvez choisir pour désigner des formulaires d'entrée de données pour une base de données.

VB

```
Public Enum InterfaceColors
    MistyRose = &HE1E4FF&
    SlateGray = &H908070&
    DodgerBlue = &HFF901E&
    DeepSkyBlue = &HFFBF00&
    SpringGreen = &H7FFF00&
    ForestGreen = &H228B22&
    Goldenrod = &H20A5DA&
    Firebrick = &H2222B2&
End Enum
```

L'exemple suivant montre des valeurs qui incluent des nombres positifs et négatifs.

VB

```
Enum SecurityLevel
    IllegalEntry = -1
    MinimumSecurity = 0
    MaximumSecurity = 1
End Enum
```

Voir aussi

Référence

Const, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Fonctions de conversion de types de données
Fonctions Asc, AscW

Concepts

Conversions implicites et explicites
Constantes et énumérations intrinsèques

Autres ressources

Constantes et énumérations en Visual Basic

Erase, instruction (Visual Basic)

Utilisée pour libérer les variables tableau et la mémoire utilisée pour leurs éléments.

```
Erase arraylist
```

Éléments
arraylist

Obligatoire. Liste de variables tableau à effacer. Les variables multiples sont séparées par des virgules.

Notes

L'instruction **Erase** peut apparaître seulement au niveau de la procédure. En d'autres termes, vous pouvez libérer des tableaux à l'intérieur d'une procédure mais pas au niveau de la classe ou du module.

L'instruction **Erase** équivaut à assigner **Nothing** à chaque variable tableau.

Exemple

L'exemple suivant utilise l'instruction **Erase** pour effacer deux tableaux et libérer leur mémoire (respectivement 1 000 et 100 éléments de stockage). L'instruction **ReDim** assigne ensuite une nouvelle instance de tableau au tableau à trois dimensions.

VB

```
Dim threeDimArray(9, 9, 9), twoDimArray(9, 9) As Integer
Erase threeDimArray, twoDimArray
ReDim threeDimArray(4, 4, 9)
```

Voir aussi
Référence
Nothing (Visual Basic)
ReDim, instruction (Visual Basic)

Error, instruction

Simule l'occurrence d'une erreur.

```
Error errornumber
```

Éléments
errornumber

Obligatoire. Peut être tout numéro d'erreur valide.

Notes

L'instruction **Error** est prise en charge pour des raisons de compatibilité descendante. Dans un nouveau code (particulièrement, lors de la création d'objets), utilisez la méthode **Raise** de l'objet **Err** pour générer des erreurs d'exécution.

Si *errornumber* est défini, l'instruction **Error** appelle le gestionnaire d'erreurs une fois que les valeurs par défaut suivantes ont été assignées aux propriétés de l'objet **Err** :

Propriété	Valeur
Number	Valeur spécifiée comme un argument de l'instruction **Error**. Peut être tout numéro d'erreur valide.
Source	Nom du projet Visual Basic en cours.
Description	Expression de chaîne correspondant à la valeur de retour de la fonction **Error** pour le **Number** spécifié, si cette chaîne existe. Si la chaîne n'existe pas, **Description** contient une chaîne de longueur nulle ("").
HelpFile	Le lecteur, le chemin d'accès et le nom de fichier complets du fichier d'aide Visual Basic approprié.
HelpContext	L'ID de contexte du fichier d'aide Visual Basic approprié pour l'erreur correspondant à la propriété **Number**.
LastDLLError	Zéro.

Si aucun gestionnaire d'erreurs n'existe ou n'est activé, un message d'erreur est créé et affiché à partir des propriétés de l'objet **Err**.

Remarque

Certaines applications hôtes Visual Basic ne peuvent pas créer d'objets. Consultez la documentation de votre application hôte pour déterminer si cette dernière peut créer des classes et des objets.

Exemple

Cet exemple utilise l'instruction **Error** pour générer l'erreur numéro 11.

```
On Error Resume Next   ' Defer error handling.
Error 11   ' Simulate the "Division by zero" error.
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic

Référence

Clear, méthode (objet Err)

Err, objet (Visual Basic)

On Error, instruction (Visual Basic)

Raise, méthode (objet Err)

Resume, instruction

Event, instruction

Déclare un événement défini par l'utilisateur.

```
[ <attrlist> ] [ accessmodifier ] _
[ Shared ] [ Shadows ] Event eventname[(parameterlist)] _
[ Implements implementslist ]
' -or-
[ <attrlist> ] [ accessmodifier ] _
[ Shared ] [ Shadows ] Event eventname As delegatename _
[ Implements implementslist ]
' -or-
 [ <attrlist> ] [ accessmodifier ] _
[ Shared ] [ Shadows ] Custom Event eventname As delegatename _
[ Implements implementslist ]
   [ <attrlist> ] AddHandler(ByVal value As delegatename)
      [ statements ]
   End AddHandler
   [ <attrlist> ] RemoveHandler(ByVal value As delegatename)
      [ statements ]
   End RemoveHandler
   [ <attrlist> ] RaiseEvent(delegatesignature)
      [ statements ]
   End RaiseEvent
End Event
```

Éléments

Élément	**Description**
attrlist	Facultatif. Liste des attributs s'appliquant à cet événement. Les attributs multiples sont séparés par des virgules. Vous devez mettre la Liste d'attributs entre des signes "inférieur à" et "supérieur à" ("<" et ">").
accessmodifier	Facultatif. Spécifie le code pouvant accéder à l'événement. Il peut s'agir de l'une des valeurs suivantes : • Public — tout code pouvant accéder à l'élément qui le déclare peut y accéder. • Protected — seul le code dans sa classe ou une classe dérivée peut y accéder. • Friend — seul le code dans le même assembly peut y accéder. • Private —- seul le code dans l'élément qui le déclare peut y accéder. Vous pouvez spécifier **Protected Friend** pour autoriser l'accès à partir du code dans la classe de l'événement, dans une classe dérivée ou dans le même assembly.
Shared	Facultatif. Spécifie que cet événement n'est pas associé à l'instance spécifique d'une classe ou d'une structure.
Shadows	Facultatif. Indique que cet événement redéclare et masque un élément de programmation de même nom ou un ensemble d'éléments surchargés, dans une classe de base. Vous pouvez occulter tout type d'élément déclaré par un autre type. Un élément occulté n'est pas disponible à partir de la classe dérivée qui l'occulte, sauf à partir de l'emplacement où l'élément d'occultation est inaccessible. Par exemple, si un élément **Private** occulte un élément de classe de base, le code qui n'est pas autorisé à accéder à l'élément **Private** accède à la place à l'élément de classe de base.
eventname	Obligatoire. Nom de l'événement. Ce nom respecte les conventions standard d'affectation de noms aux variables.

<table>
<tr><td>parameterlist</td><td>Facultatif. Liste des variables locales qui représentent les paramètres de cet événement. Vous devez mettre la Liste de paramètres entre parenthèses.</td></tr>
<tr><td>Implements</td><td>Facultatif. Indique que cet événement implémente un événement d'une interface.</td></tr>
<tr><td>implementslist</td><td>Requis si Implements est fourni. Liste des procédures Sub implémentées. Les procédures multiples sont séparées par des virgules :

implementedprocedure [, implementedprocedure ...]

Chaque implementedprocedure emploie la syntaxe et les paramètres suivants :

interface.definedname
<table>
<tr><th>Élément</th><th>Description</th></tr>
<tr><td>interface</td><td>Obligatoire. Nom d'une interface que la classe ou la structure conteneur de cette procédure implémente.</td></tr>
<tr><td>definedname</td><td>Obligatoire. Nom par lequel la procédure est définie dans interface. Il ne doit pas être identique à name, le nom que cette procédure utilise pour implémenter la procédure définie.</td></tr>
</table></td></tr>
<tr><td>Custom</td><td>Obligatoire. Les événements déclarés comme Custom doivent définir des accesseurs AddHandler, RemoveHandler et RaiseEvent personnalisés.</td></tr>
<tr><td>delegatename</td><td>Facultatif. Nom d'un délégué qui spécifie la signature du gestionnaire d'événements.</td></tr>
<tr><td>AddHandler</td><td>Obligatoire. Déclare un accesseur AddHandler, qui spécifie les instructions à exécuter lors de l'ajout d'un gestionnaire d'événements, soit explicitement à l'aide de l'instruction AddHandler soit implicitement à l'aide de la clause Handles.</td></tr>
<tr><td>End AddHandler</td><td>Obligatoire. Met fin au bloc AddHandler.</td></tr>
<tr><td>value</td><td>Obligatoire. Nom du paramètre.</td></tr>
<tr><td>RemoveHandler</td><td>Obligatoire. Déclare un accesseur RemoveHandler, qui spécifie les instructions à exécuter lors de la suppression d'un gestionnaire d'événements à l'aide de l'instruction RemoveHandler.</td></tr>
<tr><td>End RemoveHandler</td><td>Obligatoire. Met fin au bloc RemoveHandler.</td></tr>
<tr><td>RaiseEvent</td><td>Obligatoire. Déclare un accesseur RaiseEvent, qui spécifie les instructions à exécuter lors du déclenchement de l'événement à l'aide de l'instruction RaiseEvent. En général, il appelle une liste de délégués gérée par les accesseurs AddHandler et RemoveHandler.</td></tr>
<tr><td>End RaiseEvent</td><td>Obligatoire. Met fin au bloc RaiseEvent.</td></tr>
<tr><td>delegatesignature</td><td>Obligatoire. Liste des paramètres qui correspondent aux paramètres requis par le délégué delegatename. Vous devez mettre la Liste de paramètres entre parenthèses.</td></tr>
</table>

statements	Facultatif. Instructions qui contiennent les corps des méthodes **AddHandler**, **RemoveHandler** et **RaiseEvent**.
End Event	Obligatoire. Met fin au bloc **Event**.

Notes

Une fois l'événement déclaré, utilisez l'instruction **RaiseEvent** pour le déclencher. Les fragments de code suivants illustrent la déclaration et le déclenchement d'un événement type :

VB

```
Public Class EventSource
    ' Declare an event.
    Public Event LogonCompleted(ByVal UserName As String)
    Sub CauseEvent()
        ' Raise an event on successful logon.
        RaiseEvent LogonCompleted("AustinSteele")
    End Sub
End Class
```

Remarque

Vous pouvez déclarer des arguments d'événement de la même manière que pour les arguments de procédures, avec toutefois les exceptions suivantes : les événements ne peuvent pas avoir d'arguments nommés, d'arguments **ParamArray** ou d'arguments **Optional**. Les événements n'ont pas de valeurs de retour.

Pour gérer un événement, vous devez l'associer à une sous-routine du gestionnaire d'événements à l'aide de l'instruction **Handles** ou **AddHandler**. Les signatures de la sous-routine et de l'événement doivent correspondre. Pour gérer un événement partagé, vous devez utiliser l'instruction **AddHandler**.

Vous pouvez utiliser **Event** seulement au niveau du module. Cela signifie que le *contexte de déclaration* pour un événement doit être une classe, une structure, un module ou une interface, et ne peut pas être un fichier source, un espace de noms, une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Dans la plupart des cas, vous pouvez utiliser la première syntaxe présente dans la section Syntaxe de cette rubrique pour déclarer des événements. Toutefois, certains scénarios nécessitent un plus grand contrôle sur le comportement détaillé de l'événement. La dernière syntaxe présente dans la section Syntaxe de cette rubrique, qui utilise le mot clé **Custom**, fournit ce contrôle en vous permettant de définir des événements personnalisés. Dans un événement personnalisé, vous spécifiez exactement ce qui se produit lorsque le code ajoute ou supprime un gestionnaire d'événements de l'événement, ou lorsque le code déclenche l'événement. Pour obtenir des exemples, consultez
Comment : déclarer des événements qui économisent l'utilisation de la mémoire et
Comment : déclarer des événements qui évitent les blocages.

Exemple

L'exemple suivant utilise des événements pour décompter les secondes de 10 à 0. Le code illustre plusieurs méthodes, propriétés et instructions liées aux événements, notamment l'instruction **RaiseEvent**.

La classe qui déclenche un événement est la source de l'événement, et les méthodes qui traitent l'événement sont les gestionnaires d'événements. Une source d'événement peut avoir des gestionnaires multiples pour les événements qu'elle génère. Lorsque la classe génère l'événement, celui-ci se produit pour chaque classe ayant choisi de gérer les événements pour cette instance de l'objet.

Cet exemple utilise également un formulaire (`Form1`) comportant un bouton (`Button1`) et une zone de texte (`TextBox1`). Lorsque vous cliquez sur le bouton, la première zone de texte affiche un compte à rebours de 10 à 0 secondes. Lorsque la durée totale (10 secondes) s'est écoulée, la première zone de texte affiche « Terminé ».

Le code correspondant à `Form1` indique les états de début et de fin du formulaire. Il contient également le code exécuté lors du déclenchement des événements.

Pour utiliser cet exemple, ouvrez un nouveau projet Windows Forms. Ajoutez ensuite un bouton nommé `Button1` et une zone de texte nommée `TextBox1` au formulaire principal nommé `Form1`. Cliquez ensuite avec le bouton droit sur le formulaire, puis cliquez sur **Afficher le code** pour ouvrir l'éditeur de code.

Ajoutez une variable **WithEvents** à la section des déclarations de la classe `Form1` :

VB

```
Private WithEvents mText As TimerState
```

Ajoutez le code suivant au code pour `Form1`. Remplacez toute procédure en double éventuelle, telle que `Form_Load` ou `Button_Click`.

VB

```
Private Sub Form1_Load(ByVal sender As Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Button1.Text = "Start"
    mText = New TimerState
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles Button1.Click
    mText.StartCountdown(10.0, 0.1)
End Sub

Private Sub mText_ChangeText() Handles mText.Finished
    TextBox1.Text = "Done"
End Sub

Private Sub mText_UpdateTime(ByVal Countdown As Double) _
                             Handles mText.UpdateTime
    TextBox1.Text = Format(Countdown, "##0.0")
    ' Use DoEvents to allow the display to refresh.
    My.Application.DoEvents()
End Sub

Class TimerState
    Public Event UpdateTime(ByVal Countdown As Double)
    Public Event Finished()
    Public Sub StartCountdown(ByVal Duration As Double, _
                              ByVal Increment As Double)
        Dim Start As Double = DateAndTime.Timer
        Dim ElapsedTime As Double = 0

        Dim SoFar As Double = 0
        Do While ElapsedTime < Duration
            If ElapsedTime > SoFar + Increment Then
                SoFar += Increment
                RaiseEvent UpdateTime(Duration - SoFar)
            End If
            ElapsedTime = DateAndTime.Timer - Start
        Loop
        RaiseEvent Finished()
    End Sub
End Class
```

Appuyez sur F5 pour exécuter l'exemple précédent, puis cliquez sur le bouton **Démarrer**. La première zone de texte commence à décompter les secondes. Lorsque la durée totale (10 secondes) s'est écoulée, la première zone de texte affiche « Terminé ».

Remarque

La méthode **My.Application.DoEvents** ne traite pas les événements exactement de la même manière que le formulaire. Po ur permettre au formulaire de gérer directement les événements, vous pouvez utiliser le multithreading. Pour plus d'informa tions, consultez Multithreading dans Visual Basic.

Voir aussi

Tâches

Comment : ajouter des événements à une classe
Comment : déclarer des événements qui économisent l'utilisation de la mémoire
Comment : déclarer des événements qui évitent les blocages

Référence

RaiseEvent, instruction
Implements, instruction
AddHandler, instruction
RemoveHandler, instruction
Handles
Delegate, instruction
Shared (Visual Basic)
Shadows

Autres ressources

Événements en Visual Basic

Exit, instruction (Visual Basic)

Quitte une procédure ou un bloc et passe immédiatement le contrôle à l'instruction qui suit l'appel de procédure ou la définition de bloc.

```
Exit { Do | For | Function | Property | Select | Sub | Try | While }
```

Éléments

Do

Quitte immédiatement la boucle **Do** dans laquelle il apparaît. L'exécution poursuit avec l'instruction qui suit l'instruction **Loop**. L'instruction **Exit Do** peut être utilisée uniquement à l'intérieur d'une boucle **Do**. Utilisée dans des boucles **Do** imbriquées, l'instruction **Exit Do** quitte la boucle la plus profonde et transfère le contrôle au niveau d'imbrication supérieur suivant.

For

Quitte immédiatement la boucle **For** dans laquelle il apparaît. L'exécution poursuit avec l'instruction qui suit l'instruction **Next**. L'instruction **Exit For** peut être utilisée uniquement à l'intérieur d'une boucle **For**...**Next** ou **For Each**...**Next**. Utilisée dans des boucles **For** imbriquées, l'instruction **Exit For** quitte la boucle la plus profonde et transfère le contrôle au niveau d'imbrication supérieur suivant.

Function

Quitte immédiatement la procédure **Function** dans laquelle il apparaît. L'exécution poursuit avec l'instruction qui suit l'instruction qui a appelé la procédure **Function**. L'instruction **Exit Function** peut être utilisée uniquement à l'intérieur d'une procédure **Function**.

Property

Quitte immédiatement la procédure **Property** dans laquelle il apparaît. L'exécution poursuit avec l'instruction qui a appelé la procédure **Property**, c'est-à-dire l'instruction qui demande ou définit la valeur de la propriété. L'instruction **Exit Property** peut être utilisée uniquement à l'intérieur de la procédure **Get** ou **Set** d'une propriété.

Select

Quitte immédiatement le bloc **Select Case** dans lequel il apparaît. L'exécution poursuit avec l'instruction qui suit l'instruction **End Select**. L'instruction **Exit Select** peut être utilisée uniquement à l'intérieur d'une instruction **Select Case**.

Sub

Quitte immédiatement la procédure **Sub** dans laquelle il apparaît. L'exécution poursuit avec l'instruction qui suit l'instruction qui a appelé la procédure **Sub**. L'instruction **Exit Sub** peut être utilisée uniquement à l'intérieur d'une procédure **Sub**.

Try

Quitte immédiatement le bloc **Try** ou **Catch** dans lequel il apparaît. L'exécution poursuit avec le bloc **Finally** s'il en existe un, ou avec l'instruction qui suit l'instruction **End Try**, le cas échéant. L'instruction **Exit Try** peut être utilisée uniquement à l'intérieur d'un bloc **Try** ou **Catch**, mais pas à l'intérieur d'un bloc **Finally**.

While

Quitte immédiatement la boucle **While** dans laquelle il apparaît. L'exécution poursuit avec l'instruction qui suit l'instruction **End While**. L'instruction **Exit While** peut être utilisée uniquement à l'intérieur d'une boucle **While**. Utilisée à l'intérieur de boucles **While** imbriquées, l'instruction **Exit While** transfère le contrôle à la boucle située à un niveau au-dessus de la boucle dans laquelle **Exit While** apparaît.

Notes

Ne confondez pas les instructions **Exit** et les instructions **End**. **Exit** ne définit pas la fin d'une instruction.

Exemple

L'exemple suivant utilise l'instruction **Exit** pour quitter une boucle **For**...**Next**, une boucle **Do** et une procédure **Sub**.

VB

```
Sub exitStatementDemo()
    Dim demoNum As Single
    ' Set up an infinite loop.
    Do
        For i As Integer = 1 To 10000000
            demoNum = Int(Rnd() * 100)
            Select Case demoNum
                Case 7 : Exit For
                Case 29 : Exit Do
                Case 54 : Exit Sub
            End Select
        Next i
    Loop
End Sub
```

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)
End, instruction
For Each...Next, instruction (Visual Basic)
For...Next, instruction (Visual Basic)
Function, instruction (Visual Basic)
Stop, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)
For Each...Next, instruction (Visual Basic)

Instructions F-P

Le tableau suivant énumère les instructions du langage Visual Basic.

For Each...Next	For...Next	Function	Get
GoTo	If...Then...Else	Implements	Imports
Inherits	Interface	Mid	Module
Namespace	On Error	Operator	Option Compare
Option Explicit	Option Strict	Propriété	

Voir aussi

Référence

Instructions A-E

Instructions Q-Z

Autres ressources

Référence Visual Basic

For Each...Next, instruction (Visual Basic)

Répète un groupe d'instructions pour chaque élément d'une collection.

```
For Each element [ As datatype ] In group
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ element ]
```

Éléments

element

Requis dans l'instruction **For Each**. Facultatif dans l'instruction **Next**. Variable. Utilisé pour parcourir tous les éléments d'une collection.

datatype

Obligatoire si *element* n'est pas déjà déclaré. Type de données de *element*.

group

Obligatoire. Variable objet. Fait référence à la collection sur laquelle les *statements* doivent être répétées.

statements

Facultatif. Une ou plusieurs instructions entre **For Each** et **Next** qui s'exécutent sur chaque élément dans *group*.

Exit For

Facultatif. Transfert le contrôle hors de la boucle **For Each**.

Next

Obligatoire. Termine la définition de la boucle **For Each**.

Notes

Utilisez une boucle **For Each**...**Next** lorsque vous souhaitez répéter un ensemble d'instructions pour chaque élément d'une collection ou d'un tableau.

Une instruction For...Next (Visual Basic) fonctionne correctement lorsque vous pouvez associer chaque itération d'une boucle à une variable de contrôle et déterminer les valeurs initiale et finale de cette variable. Toutefois, lorsque vous gérez une collection, le concept de valeurs initiales et finales n'a pas de sens et vous n'êtes pas obligé de connaître le nombre d'éléments que contient la collection. Dans ce cas, une boucle **For Each**...**Next** est un meilleur choix.

Règles

- **Types de données.** Le type de données de *element* doit se présenter de façon telle que le type de données des éléments de *group* puisse être converti vers lui.

  Le type de données de *group* doit être un type référence qui fait référence à une collection ou à un tableau. Cela signifie que *group* doit faire référence à un objet qui implémente l'interface IEnumerable de l'espace de noms System.Collections ou l'interface IEnumerable de l'espace de noms System.Collections.Generic. **IEnumerable** définit la méthode GetEnumerator qui retourne un objet énumérateur pour la collection. L'objet énumérateur implémente l'interface IEnumerator de l'espace de noms **System.Collections** et expose la propriété Current et les méthodes Reset et MoveNext. Visual Basic les utilise pour parcourir la collection.

  Les éléments de *group* sont généralement de type **Object**, mais ils peuvent avoir n'importe quel type de données exécutable.

- **Déclaration.** Si la variable *element* n'a pas été déclarée à l'extérieur de cette boucle, vous devez la déclarer dans l'instruction **For Each**. Dans ce cas, la portée de *element* est le corps de la boucle. Cependant, vous ne pouvez pas déclarer *element* à la fois à l'extérieur et à l'intérieur de la boucle.

- **Nombre d'itérations.** Visual Basic évalue la collection une seule fois, avant que la boucle ne commence. Si votre bloc

d'instructions modifie *element* ou *group*, ces modifications n'affectent pas l'itération de la boucle.

- **Imbrication de boucles.** Vous pouvez imbriquer les boucles **For Each** en plaçant une boucle à l'intérieur d'une autre. Toutefois, chaque boucle doit posséder une variable *element* unique.

  Vous pouvez également imbriquer différents types de structures de contrôle les uns dans les autres. Pour plus d'informations, consultez Structures de contrôle imbriquées.

Remarque
Si une instruction **Next** d'un niveau d'imbrication externe est rencontrée avant le **Next** d'un niveau interne, le compilateur signale une erreur. Toutefois, le compilateur peut détecter cette erreur de chevauchement uniquement si vous spécifiez *element* dans chaque instruction **Next**.

- **Identification de la variable de contrôle.** Vous pouvez éventuellement définir *element* dans l'instruction **Next**. Cela améliore la lisibilité de votre programme, surtout si vous avez imbriqué des boucles **For Each**. Vous devez spécifier la même variable que celle qui apparaît dans l'instruction **For Each** correspondante.

- **Transfert hors de la boucle.** L'instruction Exit (Visual Basic) passe immédiatement le contrôle à l'instruction qui suit l'instruction **Next**. Vous pouvez éventuellement quitter une boucle si vous décelez une condition qui la rend inutile ou impossible pour poursuivre l'itération, telle qu'une valeur erronée ou une demande d'arrêt. Si vous décelez une exception dans une instruction **Try**...**Catch**...**Finally**, vous pouvez utiliser **Exit For** à la fin du bloc **Finally**.

  Vous pouvez placer un nombre indéterminé d'instructions **Exit For** n'importe où dans la boucle **For Each**. **Exit For** est souvent utilisé après évaluation d'une certaine condition, par exemple dans une structure **If**...**Then**...**Else**.

- **Boucles infinies.** Vous pouvez utiliser **Exit For** pour tester une condition qui engendrerait une *boucle infinie*, c'est-à-dire une boucle qui pourrait s'exécuter de nombreuses fois, voire indéfiniment. Si vous détectez une telle condition, vous pouvez utiliser **Exit For** pour abandonner la boucle. Pour plus d'informations, consultez Do...Loop, instruction (Visual Basic).

Comportement

- **Entrée dans la boucle.** Lorsque l'exécution de la boucle **For Each**...**Next** commence, Visual Basic vérifie que *group* fait référence à un objet de collection valide. Si tel n'est pas le cas, il lève une exception. Sinon, il appelle la méthode **MoveNext** et la propriété **Current** de l'objet énumérateur pour retourner le premier élément. Si **MoveNext** indique qu'il n'y a aucun élément suivant, c'est-à-dire si la collection est vide, la boucle **For Each** prend fin et le contrôle est passé à l'instruction qui suit l'instruction **Next**. Sinon, Visual Basic affecte *element* au premier élément et exécute le bloc d'instructions.

- **Itérations de la boucle.** Chaque fois que Visual Basic rencontre l'instruction **Next**, il retourne à l'instruction **For Each**. Il appelle de nouveau **MoveNext** et **Current** pou retourner l'élément suivant et exécute à nouveau le bloc ou termine la boucle selon le résultat. Ce processus continue jusqu'à ce que **MoveNext** indique qu'il n'y a aucun élément suivant ou qu'une instruction **Exit For** soit rencontrée.

- **Arrêts de la boucle.** Lorsque tous les éléments de la collection ont été successivement assignés à *element*, la boucle **For Each** se termine et le contrôle est passé à l'instruction suivant l'instruction **Next**.

- **Modification des valeurs d'itération.** Changer la valeur de *element* quand une boucle est en cours d'exécution rendra la lecture et le débogage de votre code plus difficiles. La modification de la valeur de *group* n'affecte pas la collection ou ses éléments qui étaient déterminés lorsque la boucle était entrée en premier lieu.

- **Ordre de parcours.** Lorsque vous exécutez une boucle **For Each**...**Next**, vous pouvez parcourir la collection via le contrôle de l'objet énumérateur retourné par la méthode **GetEnumerator**. L'ordre de parcours n'est pas déterminé par Visual Basic, mais par la méthode **MoveNext** de l'objet énumérateur. Ceci signifie que vous ne pourrez peut-être pas prévoir quel sera le premier élément de la collection retourné dans *element* ou quel sera l'élément suivant renvoyé après un élément donné.

  Si votre code dépend du parcours d'une collection selon un ordre défini, une boucle **For Each**...**Next** ne représente pas le meilleur choix, sauf si vous connaissez les caractéristiques de l'objet énumérateur exposé par la collection. Vous pouvez obtenir des résultats plus fiables à l'aide d'une structure de boucle différente, par exemple **For**...**Next** ou **Do**...**Loop**.

- **Modification de la collection.** L'objet énumérateur retourné par **GetEnumerator** ne vous permet normalement pas de modifier la collection par l'ajout, la suppression, le remplacement ou la réorganisation d'éléments. Si vous modifiez la collection après avoir lancé une boucle **For Each**...**Next**, l'objet énumérateur devient non valide et la tentative d'accès

suivante à un élément provoque une exception InvalidOperationException.

Toutefois, ce blocage de modification n'est pas déterminé par Visual Basic, mais plutôt par l'implémentation de l'interface **IEnumerable**. Il est possible d'implémenter **IEnumerable** de sorte que les modifications soient autorisées pendant l'itération. Si vous comptez effectuer des modifications dynamiques de ce type, soyez sûr que vous connaissez les caractéristiques de l'implémentation de **IEnumerable** sur la collection que vous utilisez.

- **Modification d'éléments de collection.** La propriété **Current** de l'objet énumérateur est ReadOnly (Visual Basic). Elle retourne une copie locale de chaque élément de la collection. Cela signifie que vous ne pouvez pas modifier les éléments d'une boucle **For Each**...**Next**. Toute modification effectuée n'a d'incidence que sur la copie locale basée sur **Current** et n'est pas répercutée dans la collection sous-jacente. Cependant, si un élément correspond à un type référence, vous pouvez modifier les membres de l'instance vers laquelle il pointe. L'exemple suivant illustre ce comportement.

  ```
  Sub lightBlueBackground(ByVal thisForm As System.Windows.Forms.Form)
      For Each thisControl As System.Windows.Forms.Control In thisForm.Controls
          thisControl.BackColor = System.Drawing.Color.LightBlue
      Next thisControl
  End Sub
  ```

  L'exemple précédent peut modifier le membre `BackColor` de chaque élément `thisControl`, bien qu'il ne puisse pas modifier `thisControl` lui-même.

- **Parcours de tableaux.** Étant donné que la classe Array implémente l'interface **IEnumerable**, tous les tableaux exposent la méthode GetEnumerator. Cela signifie que vous pouvez parcourir un tableau avec une boucle **For Each**...**Next**. Toutefois, vous pouvez uniquement lire les éléments du tableau, et non les modifier. Pour une illustration, consultez Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau.

Exemple

L'exemple suivant utilise l'instruction **For Each**...**Next** pour rechercher tous les éléments de la chaîne dans une collection "Hello". L'exemple suppose que la collection `thisCollection` a déjà été créée et que ses éléments sont de type **String**.

VB

```
Dim found As Boolean = False
Dim thisCollection As New Collection
For Each thisObject As String In thisCollection
    If thisObject = "Hello" Then
        found = True
        Exit For
    End If
Next thisObject
```

Voir aussi

Tâches

Comment : exécuter plusieurs instructions pour chaque élément dans une collection ou un tableau

Comment : améliorer les performances d'une boucle

Référence

While...End While, instruction (Visual Basic)

Do...Loop, instruction (Visual Basic)

Concepts

Structures de boucle

Collections en Visual Basic

For...Next, instruction (Visual Basic)

Répète un groupe d'instructions un nombre spécifié de fois.

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ counter ]
```

Éléments

counter

Requis dans l'instruction **For**. Variable numérique. Variable de contrôle de la boucle.

datatype

Obligatoire si *counter* n'est pas déjà déclaré. Type de données de *counter*.

start

Obligatoire. Expression numérique. Valeur initiale de *counter*.

end

Obligatoire. Expression numérique. Valeur finale de *counter*.

step

Facultatif. Expression numérique. Valeur d'incrémentation de *counter* à chaque itération de la boucle.

statements

Facultatif. Une ou plusieurs instructions entre **For** et **Next** à exécuter le nombre de fois spécifié.

Exit For

Facultatif. Transfert le contrôle hors de la boucle **For**.

Next

Obligatoire. Termine la définition de la boucle **For**.

Notes

Utilisez une structure **For**...**Next** lorsque vous souhaitez répéter une série d'instructions un nombre défini de fois.

Une boucle While...End While, instruction (Visual Basic) ou Do...Loop, instruction (Visual Basic) fonctionne correctement lorsque vous ne connaissez pas à l'avance le nombre de fois où les instructions doivent être exécutées dans une boucle. Toutefois, si vous devez exécuter la boucle un certain nombre de fois, il est préférable d'utiliser **For**...**Next**. Vous déterminez le nombre d'itérations lorsque vous entrez la boucle pour la première fois.

La valeur de *step* peut être positive ou négative. Elle détermine le traitement de la boucle comme suit :

Valeur de step	Condition d'exécution de la boucle
Positive ou zéro	*counter* <= *end*
Négative	*counter* >= *end*

Si aucune valeur n'est spécifiée, l'argument *step* prend par défaut la valeur 1.

Règles

- **Types de données.** Le type de données de *counter* est généralement **Integer**, mais il peut correspondre à tout type prenant en charge les opérateurs suivants : supérieur ou égal à (**>=**), inférieur ou égal à (**<=**), addition (**+**) et soustraction (**-**). Il peut même être un type défini par l'utilisateur à condition qu'il prenne en charge tous ces opérateurs.

Les expressions *start*, *end* et *step* ont généralement pour valeur le type **Integer**, mais peuvent avoir pour valeur tout type de données qui s'étend au type de *counter*. Si vous utilisez un type défini par l'utilisateur pour *counter*, cela signifie que vous pouvez être amené à définir l'opérateur de conversion **CType** pour convertir les types de *start*, *end* ou *step* vers le type de *counter*.

- **Déclaration.** Si la variable *counter* n'a pas été déclarée à l'extérieur de cette boucle, vous devez la déclarer dans l'instruction **For**. Dans ce cas, la portée de *counter* est le corps de la boucle. Cependant, vous ne pouvez pas déclarer *counter* à la fois à l'extérieur et à l'intérieur de la boucle.

- **Nombre d'itérations.** Visual Basic évalue les valeurs d'itération *start*, *end* et *step* une seule fois avant que la boucle ne commence. Si votre bloc d'instructions modifie *end* ou *step*, ces modifications n'affectent pas l'itération de la boucle.

- **Imbrication de boucles.** Vous pouvez imbriquer les boucles **For** en plaçant une boucle à l'intérieur d'une autre. Toutefois, chaque boucle doit posséder une variable *counter* unique. La construction suivante est valide :

```
            For i As Integer = 1 To 10
    For j As Integer = 1 To 10
        For k As Integer = 1 To 10
            ' Insert statements to operate with current values of i, j, and k.
        Next k
    Next j
Next i
```

  Vous pouvez également imbriquer différents types de structures de contrôle les uns dans les autres. Pour plus d'informations, consultez Structures de contrôle imbriquées.

  Remarque

  Si une instruction **Next** d'un niveau d'imbrication externe est rencontrée avant le **Next** d'un niveau interne, le compilateur signale une erreur. Toutefois, le compilateur peut détecter cette erreur de chevauchement uniquement si vous spécifiez *counter* dans chaque instruction **Next**.

- **Identification de la variable de contrôle.** Vous pouvez éventuellement définir *counter* dans l'instruction **Next**. Cela améliore la lisibilité de votre programme, surtout si vous avez imbriqué des boucles **For**. Vous devez spécifier la même variable que celle qui apparaît dans l'instruction **For** correspondante.

- **Transfert hors de la boucle.** L'instruction Exit, instruction (Visual Basic) passe immédiatement le contrôle à l'instruction qui suit l'instruction **Next**. Vous pouvez éventuellement quitter une boucle si vous décelez une condition qui la rend inutile ou impossible pour poursuivre l'itération, telle qu'une valeur erronée ou une demande d'arrêt. Si vous décelez une exception dans un **Try**...**Catch**...**Finally**, vous pouvez utiliser **Exit For** à la fin du bloc **Finally**.

  Vous pouvez placer un nombre indéterminé d'instructions **Exit For** n'importe où dans la boucle **For**. **Exit For** est souvent utilisé après évaluation d'une certaine condition, par exemple dans une structure **If**...**Then**...**Else**.

- **Boucles infinies.** Vous pouvez utiliser **Exit For** pour tester une condition qui engendrerait une *boucle infinie*, c'est-à-dire une boucle qui pourrait s'exécuter de nombreuses fois, voire indéfiniment. Si vous détectez une telle condition, vous pouvez utiliser **Exit For** pour abandonner la boucle. Pour plus d'informations, consultez Do...Loop, instruction (Visual Basic).

Comportement

- **Entrée dans la boucle.** Lors de l'exécution de la boucle **For**...**Next** commence, Visual Basic évalue *start*, *end* et *step* pour la seule fois. Il assigne ensuite *start* à *counter*. Avant d'exécuter le bloc d'instructions, il compare *counter* à *end*. Si *counter* est supérieur à la valeur de fin, la boucle **For** se termine et le contrôle est passé à l'instruction suivant l'instruction **Next**. Sinon, le bloc d'instructions s'exécute.

- **Itérations de la boucle.** Chaque fois que Visual Basic rencontre l'instruction **Next**, il incrémente *counter* de la valeur de *step* et retourne à l'instruction **For**. Il compare de nouveau *counter* à *end* et exécute à nouveau le bloc ou termine la boucle selon le résultat. Ce processus continue jusqu'à ce que *counter* passe *end* ou qu'une instruction **Exit For** soit rencontrée.

- **Arrêts de la boucle.** La boucle ne s'arrête pas tant que *counter* n'a pas passé *end*. Si *counter* est égal à *end*, la boucle

continue. La comparaison qui détermine l'exécution du bloc est *counter* <= *end* si *step* est positif et *counter* >= *end* si *step* est négatif.

- **Modification des valeurs d'itération.** Changer la valeur de *counter* quand une boucle est en cours d'exécution rendra la lecture et le débogage de votre code plus difficiles. Changer la valeur de *start*, *end* ou *step* n'affecte pas les valeurs d'itération déterminées lors de la première entrée de la boucle.

Exemple

L'exemple suivant illustre des structures **For**... **Next** imbriquées avec différentes valeurs d'étape.

VB

```
Dim words, digit As Integer
Dim thisString As String = ""
For words = 10 To 1 Step -1
    For digit = 0 To 9
        thisString &= CStr(digit)
    Next digit
    thisString &= " "
Next words
```

L'exemple précédent crée une chaîne qui contient 10 occurrences des chiffres de 0 à 9, les chaînes étant séparées par un espace. La boucle externe décrémente une variable de compteur de boucle à chaque itération de la boucle.

Voir aussi

Tâches

Comment : améliorer les performances d'une boucle

Référence

While...End While, instruction (Visual Basic)
Do...Loop, instruction (Visual Basic)
Exit, instruction (Visual Basic)
For Each...Next, instruction (Visual Basic)

Concepts

Structures de boucle
Structures de contrôle imbriquées

Function, instruction (Visual Basic)

Déclare le nom, les paramètres et le code qui définissent une procédure **Function**.

```
[ <attributelist> ] [ accessmodifier ] [ proceduremodifiers ] [ Shared ] [ Shadows ]
Function name [ (Of typeparamlist) ] [ (parameterlist) ] [ As returntype ] [ Implements imp
lementslist | Handles eventlist ]
    [ statements ]
    [ Exit Function ]
    [ statements ]
End Function
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

proceduremodifiers

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Overloads
- Overrides
- Overridable
- NotOverridable
- MustOverride
- **MustOverride Overrides**
- **NotOverridable Overrides**

Shared

Facultatif. Consultez Shared.

Shadows

Facultatif. Consultez Shadows.

name

Obligatoire. Nom de la procédure. Consultez Noms d'éléments déclarés.

typeparamlist

Facultatif. Liste des paramètres de type pour une procédure générique. Consultez Liste de types.

parameterlist

Facultatif. Liste des noms de variables locales qui représentent les paramètres de cette procédure. Consultez

Liste d'arguments.

returntype

Requis si **Option Strict** a la valeur **On**. Type de données de la valeur retournée par cette procédure.

Implements

Facultatif. Indique que cette procédure implémente une ou plusieurs procédures **Function**, dont chacune est définie dans une interface implémentée par la classe ou la structure conteneur de cette procédure. Consultez Implements, instruction.

implementslist

Requis si **Implements** est fourni. Liste des procédures **Function** implémentées.

```
implementedprocedure [ , implementedprocedure ... ]
```

Chaque *implementedprocedure* emploie la syntaxe et les paramètres suivants :

```
interface.definedname
```

Élément	Description
interface	Obligatoire. Nom d'une interface implémentée par la classe ou la structure conteneur de cette procédure.
definedname	Obligatoire. Nom par lequel la procédure est définie dans *interface*.

Handles

Facultatif. Indique que cette procédure peut gérer un ou plusieurs événements spécifiques. Consultez Handles.

eventlist

Requis si **Handles** est fourni. Liste des événements gérés par cette procédure.

```
eventspecifier [ , eventspecifier ... ]
```

Chaque *eventspecifier* emploie la syntaxe et les paramètres suivants :

```
eventvariable.event
```

Élément	Description
eventvariable	Obligatoire. Variable objet déclarée avec le type de données de la classe ou la structure qui déclenche l'événement.
event	Obligatoire. Nom de l'événement géré par cette procédure.

statements

Facultatif. Bloc d'instructions à exécuter à l'intérieur de cette procédure.

End Function

Met fin à la définition de cette procédure.

Notes

Tout le code exécutable doit se trouver à l'intérieur d'une procédure. Chaque procédure, à son tour, est déclarée dans une classe, structure ou module appelé la classe, structure ou module *conteneur*.

Utilisez une procédure **Function** lorsque vous devez retourner une valeur au code appelant. Utilisez une procédure **Sub** si aucune valeur ne doit être retournée.

Vous pouvez utiliser **Function** seulement au niveau du module. Cela signifie que le *contexte de déclaration* pour une fonction doit être une classe, une structure, un module ou une interface, et ne peut pas être un fichier source, un espace de noms, une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les procédures **Function** disposent par défaut d'un accès public. Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès.

Vous pouvez utiliser une procédure **Function** du côté droit d'une expression si vous voulez utiliser la valeur retournée par la fonction. Vous utilisez la procédure **Function** de la même façon que n'importe quelle fonction de bibliothèque telle que **Sqrt**,

Cos ou **ChrW**.

Vous appelez une procédure **Function** en utilisant le nom de procédure, suivi de la liste d'arguments entre parenthèses, dans une expression. Vous pouvez omettre les parenthèses seulement si vous n'indiquez aucun argument. Toutefois, votre code est plus lisible si vous incluez toujours les parenthèses.

Une fonction peut également être appelée à l'aide de l'instruction **Call**. Dans ce cas, la valeur de retour est ignorée.

Règles

- **Type de retour.** L'instruction **Function** peut déclarer le type de données de la valeur qu'elle retourne. Vous pouvez spécifier un type de données ou le nom d'une énumération, d'une structure, d'une classe ou d'une interface.

  Si vous ne spécifiez pas *returntype*, la procédure retourne **Object**.

- **Implémentation.** Si cette procédure utilise le mot clé **Implements**, la classe ou la structure conteneur doit également contenir une instruction **Implements** qui suit immédiatement son instruction **Class** ou **Structure**. L'instruction **Implements** doit contenir chaque interface spécifiée dans *implementslist*. Toutefois, le nom par lequel une interface définit **Function** (dans *definedname*) ne doit pas être identique au nom de cette procédure (dans *name*).

Comportement

- **Retour d'une procédure.** Lorsque la procédure **Function** retourne au code appelant, l'exécution se poursuit avec l'instruction qui suit celle qui l'a appelé.

  Les instructions **Exit Function** et **Return** provoquent la sortie immédiate d'une procédure **Function**. Plusieurs instructions **Exit Function** et **Return** peuvent apparaître n'importe où dans la procédure, et vous pouvez associer des instructions **Exit Function** et **Return**.

- **Valeur de retour.** Pour retourner une valeur à partir d'une fonction, vous pouvez soit assigner la valeur au nom de fonction, soit l'inclure dans une instruction **Return**. L'exemple suivant assigne la valeur de retour au nom de la fonction `myFunction`, puis utilise l'instruction **Exit Function** pour retourner :

  VB

  ```
  Function myFunction(ByVal j As Integer) As Double
      myFunction = 3.87 * j
      Exit Function
  End Function
  ```

  Si vous utilisez **Exit Function** sans assigner de valeur à *name*, la procédure retourne la valeur par défaut pour le type de données spécifié dans *returntype*. Si *returntype* n'est pas spécifié, la procédure retourne **Nothing**, la valeur par défaut pour **Object**.

  L'instruction **Return** assigne la valeur de retour et ferme la fonction simultanément. L'exemple suivant fournit une illustration.

  VB

  ```
  Function myFunction(ByVal j As Integer) As Double
      Return 3.87 * j
  End Function
  ```

Dépannage

- **Ordre d'exécution.** Visual Basic réorganise quelquefois les expressions arithmétiques de manière à optimiser l'efficacité interne. C'est pourquoi vous devez éviter d'utiliser une procédure **Function** dans une expression arithmétique quand la fonction change la valeur des variables dans la même expression.

Exemple

L'exemple suivant utilise l'instruction **Function** pour déclarer le nom, les paramètres et le code qui forment le corps d'une procédure **Function**. Le modificateur **ParamArray** permet à la fonction d'accepter un nombre variable d'arguments.

VB

```vb
Public Function calcSum(ByVal ParamArray args() As Double) As Double
    calcSum = 0
    If args.Length <= 0 Then Exit Function
    For i As Integer = 0 To UBound(args, 1)
        calcSum += args(i)
    Next i
End Function
```

L'exemple suivant appelle la fonction déclarée dans l'exemple précédent.

VB

```vb
Dim returnedValue As Double = calcSum(4, 3, 2, 1)
' The function's local variables are assigned the following values:
' args(0) = 4, args(1) = 3, and so on.
```

Voir aussi

Tâches

Comment : utiliser une classe générique

Procédures de dépannage

Référence

Sub, instruction (Visual Basic)

Liste de paramètres

Dim, instruction (Visual Basic)

Call, instruction (Visual Basic)

Of

Concepts

Tableaux de paramètres

Get, instruction

Déclare une procédure de propriété **Get** utilisée pour récupérer la valeur d'une propriété.

```
[ <attributelist> ] [ accessmodifier ] Get()
    [ statements ]
End Get
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif pour, au plus, l'une des instructions **Get** et **Set** de cette propriété. Il peut s'agir de l'une des valeurs suivantes :

- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

statements

Facultatif. Une ou plusieurs instructions qui s'exécutent lors de l'appel de la procédure de propriété **Get**.

End Get

Obligatoire. Met fin à la définition de la procédure de propriété **Get**.

Notes

Chaque propriété doit contenir une procédure de propriété **Get** à moins que la propriété soit marquée **WriteOnly**. La procédure **Get** est utilisée pour retourner la valeur actuelle de la propriété.

Visual Basic appelle automatiquement la procédure **Get** d'une propriété lorsqu'une expression demande la valeur de la propriété.

Le corps de la déclaration de propriété peut contenir uniquement les procédures **Get** et **Set** de la propriété entre l'instruction Property et **End Property**. Il ne peut stocker rien d'autre que ces procédures. En particulier, il ne peut pas stocker la valeur actuelle de la propriété. Vous devez stocker cette valeur à l'extérieur de la propriété, car si vous la stockez à l'intérieur de l'une ou l'autre des procédures de propriété, l'autre procédure de propriété ne peut pas y accéder. L'approche habituelle consiste à stocker la valeur dans une variable Private (Visual Basic) déclarée au même niveau que la propriété. Vous devez définir une procédure **Get** dans la propriété à laquelle elle s'applique.

La procédure **Get** prend par défaut le niveau d'accès de sa propriété conteneur à moins que vous utilisiez *accessmodifier* dans l'instruction **Get**.

Règles

- **Niveaux d'accès mixtes.** Si vous définissez une propriété en lecture-écriture, vous pouvez éventuellement spécifier un niveau d'accès différent pour la procédure **Get** ou **Set**, mais pas pour les deux. Dans ce cas, le niveau d'accès de la procédure doit être plus restrictif que celui de la propriété. Par exemple, si la propriété est déclarée **Friend**, vous pouvez déclarer la procédure **Get Private**, mais pas **Public**.

  Si vous définissez une propriété **ReadOnly**, la procédure **Get** représente l'ensemble de la propriété. Vous ne pouvez pas déclarer un niveau d'accès différent pour **Get**, parce que deux niveaux d'accès seront définis pour la propriété.

- **Type de retour.** L'instruction Property peut déclarer le type de données de la valeur qu'elle retourne. La procédure **Get** retourne automatiquement ce type de données. Vous pouvez spécifier un type de données ou le nom d'une énumération, d'une structure, d'une classe ou d'une interface.

Si l'instruction **Property** ne spécifie pas *returntype*, la procédure retourne **Object**.

Comportement

- **Retour d'une procédure.** Lorsque la procédure **Get** retourne au code appelant, l'exécution continue dans l'instruction qui a demandé la valeur de la propriété.

  Les procédures de propriété **Get** peuvent retourner une valeur en utilisant Return, instruction (Visual Basic) ou en assignant la valeur de retour au nom de la propriété. Pour plus d'informations, consultez « Valeur de retour » dans Function, instruction (Visual Basic).

  Les instructions **Exit Property** et **Return** provoquent la sortie immédiate d'une procédure de propriété. Plusieurs instructions **Exit Property** et **Return** peuvent apparaître n'importe où dans la procédure, et vous pouvez associer des instructions **Exit Property** et **Return**.

- **Valeur de retour.** Pour retourner une valeur à partir d'une procédure **Get**, vous pouvez soit assigner la valeur au nom de la propriété, soit l'inclure dans une instruction Return, instruction (Visual Basic). L'instruction **Return** assigne simultanément la valeur de retour de la procédure **Get** et ferme la procédure.

  Si vous utilisez **Exit Property** sans assigner de valeur au nom de la propriété, la procédure **Get** retourne la valeur par défaut pour le type de données de la propriété. Pour plus d'informations, consultez « Valeur de retour » dans Function, instruction (Visual Basic).

  L'exemple suivant illustre deux méthodes utilisées par la propriété en lecture seule `quoteForTheDay` pour retourner la valeur contenue dans la variable privée `quoteValue`.

  VB

```
Private quoteValue As String = "No quote assigned yet."
```

  VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        quoteForTheDay = quoteValue
        Exit Property
    End Get
End Property
```

  VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        Return quoteValue
    End Get
End Property
```

Exemple

L'exemple suivant utilise l'instruction **Get** pour retourner la valeur d'une propriété.

VB

```
Class propClass
    ' Define a private local variable to store the property value.
    Private currentTime As String
    ' Define the read-only property.
    Public ReadOnly Property dateAndTime() As String
        Get
            ' The Get procedure is called automatically when the
            ' value of the property is retrieved.
            currentTime = CStr(Now)
```

```
            ' Return the date and time As a string.
            Return currentTime
        End Get
    End Property
End Class
```

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Référence

Set, instruction (Visual Basic)

Property, instruction

Exit, instruction (Visual Basic)

GoTo, instruction

Effectue un branchement inconditionnel vers une ligne spécifiée d'une procédure.

```
GoTo line
```

Élément
line

Obligatoire. Toute étiquette de ligne.

Notes

L'instruction **GoTo** ne peut effectuer un branchement que vers des lignes qui appartiennent à la procédure dans laquelle elle est utilisée. La ligne doit contenir une étiquette de ligne à laquelle **GoTo** peut faire référence. Pour plus d'informations, consultez Comment : étiqueter des instructions.

Remarque

Les instructions **GoTo** peuvent rendre le code difficile à lire et à gérer. Si possible, utilisez à la place une structure de contrôle. Pour plus d'informations, consultez Flux de contrôle en Visual Basic.

Vous ne pouvez pas utiliser une instruction **GoTo** pour effectuer un branchement depuis l'extérieur d'une construction **For**...**Next**, **For Each**...**Next**, **SyncLock**...**End SyncLock**, **Try**...**Catch**...**Finally**, **With**...**End With** ou **Using**...**End Using** vers une étiquette interne.

Branchement et constructions Try

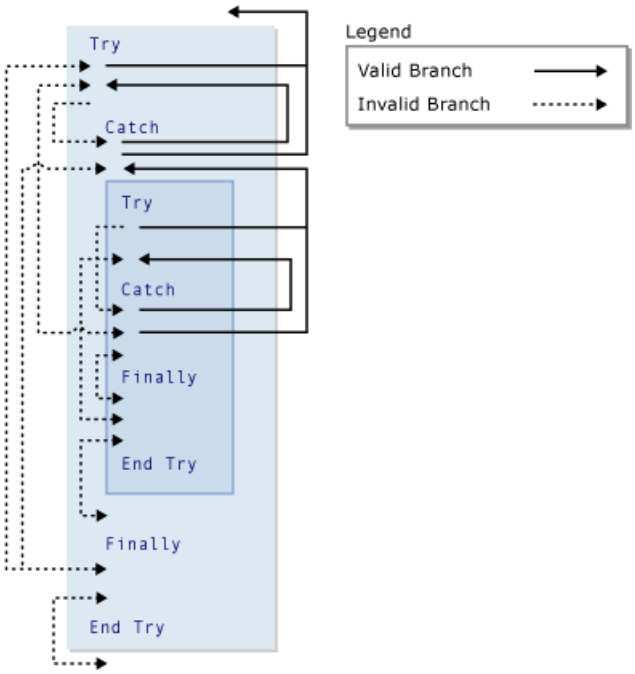
Dans une construction **Try**...**Catch**...**Finally**, les règles ci-dessous s'appliquent au branchement à l'aide de l'instruction **GoTo**.

Bloc ou région	Branchement depuis l'extérieur	Branchement depuis l'intérieur
Bloc **Try**	Uniquement à partir d'un bloc **Catch** de la même construction [1]	Uniquement vers l'extérieur de la construction entière
Bloc **Catch**	Jamais autorisé	Uniquement vers l'extérieur de la construction entière, ou vers le bloc **Try** de la même construction [1]
Bloc **Finally**	Jamais autorisé	Jamais autorisé

[1] Si une construction **Try**...**Catch**...**Finally** est imbriquée dans une autre, un bloc **Catch** peut faire l'objet d'un branchement vers le bloc **Try** à son propre niveau d'imbrication, mais pas vers un autre bloc **Try**. Une construction **Try**...**Catch**...**Finally** imbriquée doit être entièrement contenue dans un bloc **Try** ou **Catch** de la construction dans laquelle elle est imbriquée.

L'illustration ci-dessous présente une construction **Try** imbriquée dans une autre. Les divers branchements entre les blocs des deux constructions sont indiqués comme étant corrects ou incorrects.

Branchements corrects et incorrects dans les constructions Try



Exemple

L'exemple ci-dessous utilise l'instruction **GoTo** pour effectuer un branchement vers les étiquettes de ligne au sein d'une procédure.

VB

```
    Sub gotoStatementDemo()
        Dim number As Integer = 1
        Dim sampleString As String
        ' Evaluate number and branch to appropriate label.
        If number = 1 Then GoTo Line1 Else GoTo Line2
Line1:
        sampleString = "Number equals 1"
        GoTo LastLine
Line2:
        ' The following statement never gets executed because number = 1.
        sampleString = "Number equals 2"
LastLine:
        ' Write "Number equals 1" in the Debug window.
        Debug.WriteLine(sampleString)
    End Sub
```

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)
For...Next, instruction (Visual Basic)
For Each...Next, instruction (Visual Basic)
If...Then...Else, instruction (Visual Basic)
Select...Case, instruction (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)
While...End While, instruction (Visual Basic)
With...End With, instruction (Visual Basic)

If...Then...Else, instruction (Visual Basic)

Exécute un groupe d'instructions soumises à une condition, en fonction de la valeur d'une expression.

```
If condition [ Then ]
    [ statements ]
[ ElseIf elseifcondition [ Then ]
    [ elseifstatements ] ]
[ Else
    [ elsestatements ] ]
End If
-or-
If condition Then [ statements ] [ Else [ elsestatements ] ]
```

Éléments

condition

Obligatoire. Expression. Doit avoir pour valeur **True** ou **False**, ou à un type de données qui est implicitement convertible en **Boolean**.

Then

Obligatoire dans la syntaxe à une seule ligne, facultatif dans la syntaxe multiligne.

statements

Facultatif. Une ou plusieurs instructions qui suivent **If**...**Then** sont exécutées si *condition* a la valeur **True**.

elseifcondition

Obligatoire si **ElseIf** est présent. Expression. Doit avoir pour valeur **True** ou **False**, ou à un type de données qui est implicitement convertible en **Boolean**.

elseifstatements

Facultatif. Une ou plusieurs instructions qui suivent **ElseIf**...**Then** sont exécutées si *elseifcondition* a la valeur **True**.

elsestatements

Facultatif. Une ou plusieurs instructions sont exécutées si aucune expression *condition* ou *elseifcondition* précédente n'a la valeur **True**.

End If

Met fin au bloc **If**...**Then**...**Else**.

Notes

Vous pouvez utiliser la syntaxe à une seule ligne pour effectuer de petits tests simples. Toutefois, la syntaxe multiligne fournit une plus grande structure et davantage de souplesse. En outre, elle est généralement plus facile à lire, à gérer et à déboguer.

En présence d'une instruction **If**...**Then**...**Else** multiligne, *condition* est testé. Si *condition* a la valeur **True**, les instructions qui suivent **Then** sont exécutées. Si *condition* a la valeur **False**, chaque instruction **ElseIf** est évaluée à tour de rôle. Quand une *elseifcondition* de valeur **True** est trouvée, les instructions qui suivent immédiatement l'instruction **Then** correspondante sont exécutées. Si aucune *elseifcondition* n'a la valeur **True**, ou s'il n'existe aucune instruction **ElseIf**, les instructions qui suivent **Else** sont exécutées. Après l'exécution des instructions qui suivent **Then**, **ElseIf** ou **Else**, l'exécution se poursuit avec l'instruction qui suit **End If**.

Conseil

Select...Case, instruction (Visual Basic) peut s'avérer mieux adapté à l'évaluation d'une seule expression qui possède plusieurs valeurs possibles.

Dans le cas de la syntaxe à une seule ligne, il est possible d'exécuter plusieurs instructions à partir d'une décision **If**...**Then**. Toutes les instructions doivent être présentes sur la même ligne et être séparées par le signe deux-points. C'est ce qu'illustre l'exemple suivant.

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

Dans la syntaxe multiligne, l'instruction **If** doit être la seule instruction présente sur la première ligne. Les instructions **ElseIf**, **Else** et **End If** ne peuvent être précédées que par une étiquette de ligne. L'instruction **If**...**Then**...**Else** multiligne doit se terminer par une instruction **End If**.

Pour déterminer si une instruction **If** introduit une syntaxe multiligne, examinez ce qui suit le mot clé **Then**. Si tout élément autre qu'un commentaire apparaît après **Then** dans la même instruction, l'instruction est traitée comme une instruction **If** à une seule ligne. Si **Then** est absent, il s'agit du début d'une instruction **If**...**Then**...**Else** multiligne.

Les clauses **ElseIf** et **Else** sont toutes deux facultatives. Une instruction **If**...**Then**...**Else** multiligne peut contenir autant de clauses **ElseIf** que vous le souhaitez, mais aucune ne peut apparaître après une clause **Else**. Les syntaxes multilignes peuvent être imbriquées les unes aux autres.

Exemple

L'exemple suivant illustre l'instruction **If**...**Then**...**Else** sous la forme d'une ligne unique ou de plusieurs lignes.

VB

```
Dim number, digits As Integer
Dim myString As String
number = 53
If number < 10 Then
    digits = 1
ElseIf number < 100 Then
    digits = 2
Else
    digits = 3
End If
If digits = 1 Then myString = "One" Else myString = "More than one"
```

Dans l'exemple précédent, la condition **ElseIf** a la valeur **True**, et la valeur 2 est assignée à `digits`. La dernière instruction assigne ensuite la valeur "Plusieurs" à `myString`.

Voir aussi

Référence

#If...Then...#Else, directives
Choose, fonction
Select...Case, instruction (Visual Basic)
Switch, fonction

Implements, instruction

Spécifie une ou plusieurs interfaces, ou des membres d'interface qui doivent être implémentés dans la définition de classe ou de structure dans laquelle elle apparaît.

```
Implements interfacename [, ...]
-or-
Implements interfacename.interfacemember [, ...]
```

Éléments

interfacename

Obligatoire. Une interface dont les propriétés, procédures et événements doivent être implémentés par des membres correspondants dans la classe ou la structure.

interfacemember

Obligatoire. Le membre d'une interface en cours d'implémentation.

Notes

Une interface est une collection de prototypes représentant les membres (propriétés, procédures et événements) que l'interface encapsule. Les interfaces contiennent uniquement les déclarations des membres ; les classes et les structures implémentent ces membres.

L'instruction **Implements** doit suivre immédiatement l'instruction **Class** ou **Structure**.

Lorsque vous implémentez une interface, vous devez implémenter tous les membres déclarés dans l'interface. L'omission d'un membre est considérée comme une erreur de syntaxe. Pour implémenter un membre, vous spécifiez le mot clé Implements (Visual Basic) (qui est distinct de l'instruction **Implements**) lorsque vous déclarez le membre dans la classe ou la structure. Pour plus d'informations, consultez Implements, mot clé et instruction.

Les classes peuvent utiliser des implémentations Private (Visual Basic) des propriétés et procédures, mais ces membres sont uniquement accessibles par le casting d'une instance de la classe d'implémentation en une variable déclarée comme étant du type de l'interface.

Exemple

L'exemple suivant illustre l'utilisation de l'instruction **Implements** pour implémenter les membres d'une interface. Il définit une interface nommée `ICustomerInfo` avec un événement, une propriété et une procédure. La classe `customerInfo` implémente tous les membres définis dans l'interface.

VB

```
Public Interface ICustomerInfo
    Event updateComplete()
    Property customerName() As String
    Sub updateCustomerStatus()
End Interface

Public Class customerInfo
    Implements ICustomerInfo
    ' Storage for the property value.
    Private customerNameValue As String
    Public Event updateComplete() Implements ICustomerInfo.updateComplete
    Public Property CustomerName() As String Implements _
        ICustomerInfo.customerName
        Get
            Return customerNameValue
        End Get
        Set(ByVal value As String)
            ' The value parameter is passed to the Set procedure
            ' when the contents of this property are modified.
            customerNameValue = value
        End Set
    End Property
```

```
    Public Sub updateCustomerStatus() Implements _
        ICustomerInfo.updateCustomerStatus
        ' Add code here to update the status of this account.
        ' Raise an event to indicate that this procedure is done.
        RaiseEvent updateComplete()
    End Sub
End Class
```

Notez que la classe `customerInfo` utilise l'instruction **Implements** sur une ligne de code source distincte pour indiquer que la classe implémente tous les membres de l'interface `ICustomerInfo`. Ensuite, chaque membre de la classe utilise le mot clé **Implements** dans sa déclaration membre pour indiquer qu'il implémente ce membre d'interface.

Les deux procédures suivantes illustrent l'utilisation de l'interface implémentée dans l'exemple précédent. Pour tester l'implémentation, ajoutez ces deux procédures à votre projet et appelez la procédure `testImplements`.

VB

```
Public Sub testImplements()
    ' This procedure tests the interface implementation by
    ' creating an instance of the class that implements ICustomerInfo.
    Dim cust As ICustomerInfo = New customerInfo()
    ' Associate an event handler with the event that is raised by
    ' the cust object.
    AddHandler cust.updateComplete, AddressOf handleUpdateComplete
    ' Set the customerName Property
    cust.customerName = "Fred"
    ' Retrieve and display the customerName property.
    MsgBox("Customer name is: " & cust.customerName)
    ' Call the updateCustomerStatus procedure, which raises the
    ' updateComplete event.
    cust.updateCustomerStatus()
End Sub

Sub handleUpdateComplete()
    ' This is the event handler for the updateComplete event.
    MsgBox("Update is complete.")
End Sub
```

Voir aussi

Référence

Implements (Visual Basic)

Interface, instruction (Visual Basic)

Concepts

Polymorphisme dans Visual Basic

Autres ressources

Interfaces en Visual Basic

Polymorphisme

Imports, instruction

Importe des espaces de noms ou des éléments de programmation définis dans les projets et les assemblys référencés. Importe également des espaces de noms ou des éléments définis dans le même projet.

```
Imports [ aliasname = ] namespace
-or-
Imports [ aliasname = ] namespace.element
```

Éléments

aliasname

Facultatif. *Alias d'importation* ou nom par lequel le code peut faire référence à *namespace* au lieu de la chaîne de qualification complète. Consultez Noms d'éléments déclarés.

namespace

Obligatoire. Nom qualifié complet de l'espace de noms importé. Il peut s'agir d'une chaîne d'espaces de noms imbriqués à un niveau quelconque.

element

Facultatif. Nom d'un élément de programmation déclaré dans l'espace de noms. Il peut s'agir de n'importe quel élément conteneur.

Notes

Chaque fichier source peut contenir un nombre quelconque d'instructions **Imports**. Celles-ci doivent suivre les déclarations d'option telles que l'instruction **Option Strict**, et elles doivent précéder les déclarations d'élément de programmation telles que les instructions **Module** ou **Class**.

Vous pouvez utiliser **Imports** seulement au niveau du fichier. Cela signifie que le contexte de déclaration pour l'importation doit être un fichier source, et ne peut pas être un espace de noms, une classe, une structure, un module, une interface, une procédure ou un bloc.

Les alias d'importation sont utiles lorsque vous devez utiliser des éléments portant le même nom et qui sont déclarés dans un ou plusieurs espaces de noms. Pour plus d'informations et pour obtenir un exemple, consultez « Classes avec le même nom » dans Résolution d'une référence lorsque plusieurs variables ont le même nom.

Notez que l'instruction **Imports** ne rend pas d'éléments d'autres projets et assemblys disponibles à votre projet. L'importation ne prend pas la place de la définition d'une référence. Elle rend inutile la qualification des noms qui sont déjà disponibles pour votre projet. Pour plus d'informations, consultez « Importation d'éléments conteneurs » dans Résolution d'une référence lorsque plusieurs variables ont le même nom.

Règles

- **Nom de l'alias.** Vous ne devez pas déclarer de membre au niveau du module avec le même nom que *aliasname*. Sinon, le compilateur Visual Basic utilise uniquement *aliasname* pour le membre déclaré et ne le reconnaît plus comme alias d'importation.
- **Nom de l'espace de noms.** Vous pouvez fournir un nom d'espace de noms unique ou une chaîne d'espaces de noms imbriqués. Chaque espace de noms imbriqué est séparé de l'espace de noms du niveau supérieur suivant par un point (**.**), comme l'exemple suivant l'illustre.

  ```
  Imports System.Collections.Generic
  ```

- **Type d'élément.** Si vous fournissez *element*, il doit représenter un *élément conteneur*, c'est-à-dire un élément de programmation qui peut contenir d'autres éléments. Les éléments conteneurs comprennent des classes, structures, modules, interfaces et énumérations.

Comportement

- **Portée.** La portée des éléments mis à disposition par une instruction **Imports** varie selon que vous spécifiez *element*. Si vous spécifiez uniquement *namespace*, tous les membres à nom unique de cet espace de noms ainsi que les membres des éléments conteneurs présents dans cet espace de noms, sont disponibles sans qualification. Si vous spécifiez

namespace et *element*, seuls les membres de cet élément sont disponibles sans qualification.

- **Qualification.** Le code présent à l'extérieur d'un espace de noms ou d'un élément conteneur doit qualifier normalement le nom d'un membre avec le nom de cet espace de noms de l'élément conteneur. L'instruction **Imports** rend cette qualification inutile à moins que votre projet ait accès à un autre membre dont le nom est identique. Dans ce cas, vous pouvez spécifier *aliasname* dans chaque instruction **Imports**. Ensuite, vous n'avez besoin que des alias d'importation pour qualifier les membres portant le même nom.

Exemple

L'exemple suivant importe la classe Microsoft.VisualBasic.Strings et assigne un alias, `str`, pouvant être utilisé pour accéder à la méthode Left.

VB

```
' Place Imports statements at the top of your program.
Imports str = Microsoft.VisualBasic.Strings
```

VB

```
Class testClass1
    Sub showHello()
        ' Display only the word "Hello"
        MsgBox(str.Left("Hello World", 5))
    End Sub
End Class
```

Notez que l'exemple précédent importe un espace de noms imbriqué, `Strings` dans `VisualBasic` dans `Microsoft`. Lorsque MsgBox, fonction (Visual Basic) accède à la méthode **Left**, il peut utiliser l'alias `str` au lieu de l'ensemble de la chaîne de qualification `Microsoft.VisualBasic.Strings`.

Voir aussi

Référence

Namespace, instruction

Concepts

Introduction à la bibliothèque de classes .NET Framework dans Visual Studio

Espaces de noms dans Visual Basic

Autres ressources

Visual Basic et le .NET Framework

Inherits, instruction

La classe ou l'interface en cours hérite des attributs, variables, propriétés, procédures et événements d'une autre classe ou d'un autre ensemble d'interfaces.

```
Inherits basetypenames
```

Éléments
basetypenames

Obligatoire. Nom de la classe d'où dérive cette classe.

- ou -

Noms des interfaces d'où dérive cette interface. Utilisez des virgules pour séparer plusieurs noms.

Notes

Si elle est utilisée, l'instruction **Inherits** doit être la première ligne non vide et sans commentaire dans une définition de classe ou d'interface. Elle doit suivre immédiatement l'instruction **Class** ou **Interface**.

Vous pouvez utiliser **Inherits** uniquement dans une classe ou une interface. Cela signifie que le contexte de déclaration pour un héritage ne peut pas être un fichier source, un espace de noms, une structure, un module, une procédure ou un bloc.

Règles

- **Héritage de la classe.** Si une classe utilise l'instruction **Inherits**, vous ne pouvez spécifier qu'une seule classe de base.

  Une classe ne peut pas hériter d'une classe qui y est imbriquée.

- **Héritage de l'interface.** Si une interface utilise l'instruction **Inherits**, vous pouvez spécifier une ou plusieurs interfaces de base. Vous pouvez hériter de deux interfaces même si chacune définit un membre avec le même nom. Dans ce cas, le code d'implémentation doit utiliser la qualification de nom pour spécifier le membre qu'il implémente.

  Une interface ne peut pas hériter d'une autre interface dont le niveau d'accès est plus restrictif. Par exemple, une interface **Public** ne peut pas hériter d'une interface **Friend**.

  Une interface ne peut pas hériter d'une interface qui y est imbriquée.

Un exemple d'héritage de classe dans le .NET Framework est la classe ArgumentException qui hérite de la classe SystemException. Elle fournit à **ArgumentException** toutes les propriétés et procédures prédéfinies requises par les exceptions système, telles que la propriété Message et la méthode ToString.

Un exemple d'héritage d'interface dans le .NET Framework est l'interface ICollection qui hérite de l'interface IEnumerable. **ICollection** hérite de la définition de l'énumérateur permettant de parcourir une collection.

Exemple

L'exemple suivant utilise l'instruction **Inherits** pour montrer comment une classe nommée `thisClass` peut hériter de tous les membres d'une classe de base nommée `anotherClass`.

VB

```
Public Class thisClass
    Inherits anotherClass
    ' Add code to override, overload, or extend members
    ' inherited from the base class.
    ' Add new variable, property, procedure, and event declarations.
End Class
```

L'exemple suivant présente l'héritage de plusieurs interfaces.

VB

```
Public Interface thisInterface
```

```
    Inherits IComparable, IDisposable, IFormattable
    ' Add new property, procedure, and event definitions.
End Interface
```

L'interface nommée `thisInterface` contient maintenant toutes les définitions des interfaces IComparable, IDisposable et IFormattable. Les membres hérités fournissent respectivement la comparaison spécifique au type de deux objets, en libérant des ressources allouées et en exprimant la valeur d'un objet en tant que **String**. Une classe qui implémente `thisInterface` doit implémenter chaque membre de chaque interface de base.

Voir aussi

Référence

MustInherit

NotInheritable

MyBase

MyClass

Concepts

Exemples d'implémentation d'interface dans Visual Basic

Autres ressources

Héritage en Visual Basic

Interface, instruction (Visual Basic)

Déclare le nom d'une interface et introduit les définitions des membres contenus dans l'interface.

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] _
Interface name [ ( Of typelist ) ]
    [ Inherits interfacenames ]
    [ [ modifiers ] Property membername ]
    [ [ modifiers ] Function memberame ]
    [ [ modifiers ] Sub memberame ]
    [ [ modifiers ] Event memberame ]
    [ [ modifiers ] Interface memberame ]
    [ [ modifiers ] Class memberame ]
    [ [ modifiers ] Structure memberame ]
End Interface
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

Shadows

Facultatif. Consultez Shadows.

name

Obligatoire. Nom de cette interface. Consultez Noms d'éléments déclarés.

Of

Facultatif. Indique qu'il s'agit d'une interface générique.

typelist

Obligatoire si vous utilisez le mot clé Of. Liste de paramètres de type pour cette interface. Consultez Liste de types.

Inherits

Facultatif. Indique que cette interface hérite des attributs et des membres d'une autre ou d'autres interfaces. Consultez Inherits, instruction.

interfacenames

Obligatoire si vous utilisez l'instruction **Inherits**. Noms des interfaces d'où dérive cette interface.

modifiers

Facultatif. Modificateurs appropriés pour le membre d'interface défini.

Property

Facultatif. Définit une propriété qui est membre de l'interface.

Function

Facultatif. Définit une procédure **Function** qui est membre de l'interface.

Sub

Facultatif. Définit une procédure **Sub** qui est membre de l'interface.

Event

Facultatif. Définit un événement qui est membre de l'interface.

Interface

Facultatif. Définit une interface qui est imbriquée dans cette interface. La définition de l'interface imbriquée doit se terminer par une instruction **End Interface**.

Class

Facultatif. Définit une classe qui est membre de l'interface. La définition de la classe membre doit se terminer par une instruction **End Class**.

Structure

Facultatif. Définit une structure qui est membre de l'interface. La définition de la structure membre doit se terminer par une instruction **End Structure**.

membername

Obligatoire pour chaque propriété, procédure, événement, interface, classe ou structure défini comme membre de l'interface. Nom du membre.

End Interface

Met fin à la définition **Interface**.

Notes

Une *interface* définit un ensemble de membres, par exemple des propriétés et des procédures que les classes et les structures peuvent implémenter. L'interface définit uniquement les signatures des membres, mais pas leurs mécanismes internes.

Une classe ou une structure implémente l'interface en fournissant un code pour chaque membre défini par l'interface. Enfin, lorsque l'application crée une instance à partir de cette classe ou structure, un objet existe et s'exécute en mémoire. Pour plus d'informations, consultez Programmation orientée objet dans Visual Basic et Interfaces en Visual Basic.

Pour une comparaison des interfaces et des classes, consultez Vue d'ensemble des interfaces.

Vous pouvez utiliser **Interface** uniquement au niveau de l'espace de noms ou du module. Cela signifie que le *contexte de déclaration* pour une interface doit être un fichier source, un espace de noms, une classe, une structure, un module ou une interface, et ne peut pas être une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les interfaces disposent par défaut d'un accès Friend (Visual Basic). Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Règles

- **Imbrication des interfaces.** Vous pouvez définir une interface dans une autre. L'interface externe est appelée *interface conteneur*, et l'interface interne est appelée *interface imbriquée*.
- **Déclaration de membre.** Lorsque vous déclarez une propriété ou une procédure comme membre d'une interface, vous définissez uniquement la *signature* de cette propriété ou procédure. Cela inclut le type d'élément (propriété ou procédure), ses paramètres et types de paramètre et son type de retour. De ce fait, la définition du membre utilise une seule ligne de code, et les instructions de fin telles que **End Function** ou **End Property** ne sont pas valides dans une interface.

  En revanche, lorsque vous définissez une énumération ou une structure, ou encore une classe ou une interface imbriquée, il est nécessaire d'inclure leurs données membres.
- **Modificateurs de membre.** Vous ne pouvez pas utiliser de modificateurs d'accès lors de la définition de membres de module, et vous ne pouvez pas spécifier Shared (Visual Basic) ou un modificateur de procédure sauf Overloads. Vous pouvez déclarer tout membre avec Shadows, et vous pouvez utiliser Default (Visual Basic), ainsi que

ReadOnly (Visual Basic) ou WriteOnly, lors de la définition d'une propriété.

- **Héritage.** Si l'interface utilise l'Inherits, instruction, vous pouvez spécifier une ou plusieurs interfaces de base. Vous pouvez hériter de deux interfaces même si chacune définit un membre avec le même nom. Dans ce cas, le code d'implémentation doit utiliser la qualification de nom pour spécifier le membre qu'il implémente.

  Une interface ne peut pas hériter d'une autre interface dont le niveau d'accès est plus restrictif. Par exemple, une interface **Public** ne peut pas hériter d'une interface **Friend**.

  Une interface ne peut pas hériter d'une interface qui y est imbriquée.

- **Implémentation.** Lorsqu'une classe utilise l'instruction Implements (Visual Basic) pour implémenter cette interface, elle doit implémenter chaque membre défini dans l'interface. En outre, chaque signature dans le code d'implémentation doit correspondre exactement à la signature associée définie dans cette interface. Toutefois, le nom du membre dans le code d'implémentation ne doit pas correspondre au nom du membre défini dans l'interface.

  Lorsqu'une classe implémente une procédure, elle ne peut pas désigner la procédure comme **Shared**.

- **Propriété par défaut.** Une interface peut spécifier au plus une propriété comme sa *propriété par défaut*, qui peut être référencée sans utiliser le nom de la propriété. Vous spécifiez cette propriété en la déclarant avec le modificateur Default (Visual Basic). Pour plus d'informations, consultez Propriétés par défaut.

  Cela signifie qu'une interface peut définir une propriété par défaut uniquement si elle n'en hérite d'aucune.

Comportement

- **Niveau d'accès.** Tous les membres d'interface possèdent implicitement un accès Public (Visual Basic). Vous ne pouvez pas utiliser un modificateur d'accès lors de la définition d'un membre. Toutefois, une classe qui implémente l'interface peut déclarer un niveau d'accès pour chaque membre implémenté.

  Si vous assignez une instance de classe à une variable, le niveau d'accès de ses membres varie selon que le type de données de la variable est l'interface sous-jacente ou la classe d'implémentation. L'exemple suivant illustre ce comportement :

  VB

```
Public Interface IDemo
    Sub doSomething()
End Interface
Public Class implementIDemo
    Implements IDemo
    Private Sub doSomething() Implements IDemo.doSomething
    End Sub
End Class
Dim varAsInterface As IDemo = New implementIDemo()
Dim varAsClass As implementIDemo = New implementIDemo()
```

  Si vous accédez aux membres d'une classe par l'intermédiaire de `varAsInterface`, ils disposent tous d'un accès public. Toutefois, si vous accédez aux membres par l'intermédiaire de `varAsClass`, la procédure **Sub** `doSomething` dispose d'un accès privé.

- **Portée.** Une interface est présente dans la portée dans l'ensemble de son espace de noms, sa classe, sa structure ou son module.

  La portée de chaque membre d'interface est l'interface entière.

- **Durée de vie.** Une interface proprement dite n'a pas de durée de vie, ni ses membres. Lorsqu'une classe implémente une interface et qu'un objet est créé en tant qu'instance de cette classe, l'objet a une durée de vie dans l'application dans laquelle il s'exécute. Pour plus d'informations, consultez « Durée de vie » dans Class, instruction (Visual Basic).

Exemple

L'exemple suivant utilise l'instruction **Interface** pour définir une interface nommée `thisInterface`, qui doit être implémentée avec une instruction **Property** et une instruction **Function**.

VB

```
Public Interface thisInterface
    Property thisProp(ByVal thisStr As String) As Char
    Function thisFunc(ByVal thisInt As Integer) As Integer
End Interface
```

Notez que les instructions **Property** et **Function** n'introduisent pas de blocs se terminant par **End Property** et **End Function** dans l'interface. L'interface définit uniquement les signatures de ses membres. Les blocs **Property** et **Function** complets apparaissent dans une classe qui implémente `thisInterface`.

Voir aussi

Référence

Class, instruction (Visual Basic)
Module, instruction
Structure, instruction
Property, instruction
Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)

Concepts

Types génériques en Visual Basic

Autres ressources

Interfaces en Visual Basic

Mid, instruction

Remplace un nombre spécifié de caractères d'une variable **String** par les caractères d'une autre chaîne.

```
Mid( _
   ByRef Target As String, _
   ByVal Start As Integer, _
   Optional ByVal Length As Integer _
) = StringExpression
```

Éléments

Target

Requis. Nom de la variable **String** à modifier.

Start

Requis. Expression **Integer**. Position du caractère dans *Target* à l'endroit où le remplacement du texte commence. *Start* utilise un index de base 1.

Length

Facultatif. Expression **Integer**. Nombre de caractères à remplacer. Si cette valeur est omise, *String* est utilisé dans son intégralité.

StringExpression

Requis. Expression **String** qui remplace une partie de *Target*.

Exceptions

Type d'exception	Numéro de l'erreur	Condition
ArgumentException	5	*Start* <= 0 ou *Length* < 0.

Consultez la colonne « Numéro d'erreur » si vous mettez à niveau des applications Visual Basic 6.0 qui utilisent la gestion des erreurs non structurée. (Vous pouvez comparer le numéro d'erreur par rapport à Number, propriété (objet Err).) Toutefois, lorsque cela est possible, vous devez envisager de remplacer un tel contrôle d'erreur par Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic.

Notes

Le nombre de caractères remplacé est toujours inférieur ou égal au nombre de caractères dans *Target*.

Visual Basic possède les instructions Mid, fonction (Visual Basic) et **Mid**. Ces éléments fonctionnent à la fois sur un nombre spécifié de caractères dans une chaîne, mais la fonction **Mid** retourne les caractères pendant que l'instruction **Mid** remplace les caractères. Pour plus d'informations, consultez Mid, fonction (Visual Basic).

Remarque

Dans les précédentes versions de Visual Basic, l'instruction **MidB** remplace une sous-chaîne en octets, plutôt que des caractères. Son rôle est d'abord de convertir des chaînes en applications à jeu de caractères codés sur deux octets (DBCS, Double-Byte Character Set). Toutes les chaînes Visual Basic sont en Unicode, et la fonction **MidB** n'est plus prise en charge.

Exemple

L'exemple suivant utilise l'instruction **Mid** pour remplacer un nombre spécifié de caractères dans une variable chaîne par les caractères d'une autre chaîne.

VB

```
Dim TestString As String
' Initializes string.
TestString = "The dog jumps"
' Returns "The fox jumps".
Mid(TestString, 5, 3) = "fox"
```

```
' Returns "The cow jumps".
Mid(TestString, 5) = "cow"
' Returns "The cow jumpe".
Mid(TestString, 5) = "cow jumped over"
' Returns "The duc jumpe".
Mid(TestString, 5, 3) = "duck"
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Strings

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Mid, fonction (Visual Basic)

Autres ressources

Chaînes en Visual Basic

Introduction aux chaînes dans Visual Basic

Module, instruction

Déclare le nom d'un module et introduit la définition des variables, propriétés, événements et procédures contenus dans le module.

```
[ <attributelist> ] [ accessmodifier ]  Module name
    [ statements ]
End Module
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Friend

Consultez Niveaux d'accès dans Visual Basic.

name

Obligatoire. Nom de ce module. Consultez Noms d'éléments déclarés.

statements

Facultatif. Instructions qui définissent les variables, propriétés, événements, procédures et types imbriqués de ce module.

End Module

Met fin à la définition **Module**.

Notes

Une instruction **Module** définit un type référence disponible dans l'ensemble de son espace de noms. Un *module* (parfois appelé *module standard*) est semblable à une classe, à quelques distinctions importantes près. Chaque module contient exactement une instance et ne doit pas être créé ou assigné à une variable. Les modules ne prennent pas en charge l'héritage ou n'implémentent pas d'interfaces. Notez qu'un module n'est pas un *type* dans le sens où une classe ou une structure l'est ; vous ne pouvez pas déclarer un élément de programmation comme ayant le type de données d'un module.

Vous pouvez utiliser **Module** seulement au niveau de l'espace de noms. Cela signifie que le *contexte de déclaration* pour un module doit être un fichier source ou un espace de noms, et ne peut pas être une classe, une structure, un module, une interface, une procédure ou un bloc. Vous ne pouvez pas imbriquer un module dans un autre module, ou dans un type. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Un module a la même durée de vie que votre programme. Ses membres étant tous **Shared**, ils ont également une durée de vie équivalente à celle du programme.

Les modules disposent par défaut d'un accès Friend (Visual Basic). Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Tous les membres d'un module sont implicitement **Shared**.

Classes et modules

Ces éléments présentent plusieurs similitudes, mais également des différences importantes.

- **Terminologie.** Les versions antérieures de Visual Basic reconnaissent deux types de modules : les *modules de classe* (fichiers .cls) et les *modules standard* (fichiers .bas). Dans la version actuelle, ils sont désignés par *classes* et *modules*, respectivement.
- **Membres partagés.** Vous pouvez contrôler si un membre d'une classe est un membre partagé ou un membre d'instance.

- **Orientation de l'objet.** Les classes sont orientées objet, mais les modules ne le sont pas. Par conséquent, seules les classes peuvent être instanciées comme des objets. Pour plus d'informations, consultez Classes et modules.

Règles

- **Modificateurs.** Tous les membres de module sont implicitement Shared (Visual Basic). Vous ne pouvez pas utiliser le mot clé **Shared** lors de la déclaration d'un membre, et vous ne pouvez pas modifier l'état partagé d'un membre.
- **Héritage.** Un module ne peut pas hériter d'un type autre que Object, d'où héritent tous les modules. En particulier, un module ne peut pas hériter d'un autre.

  Vous ne pouvez pas utiliser l'instruction Inherits dans une définition de module, même pour spécifier **Object**.
- **Propriété par défaut.** Vous ne pouvez pas définir les propriétés Propriétés par défaut dans un module.

Comportement

- **Niveau d'accès.** Dans un module, vous pouvez déclarer chaque membre avec son propre niveau d'accès. Les membres de module disposent par défaut d'un accès Public (Visual Basic), sauf les variables et les constantes qui possèdent par défaut un accès Private (Visual Basic). Lorsqu'un module dispose d'un accès plus restreint que l'un de ses membres, le niveau d'accès du module spécifié est prioritaire.
- **Portée.** Un module est présent dans la portée pour son espace de noms.

  La portée de chaque membre de module est le module entier. Notez que tous les membres subissent une *promotion de type* qui entraîne la promotion de leur portée dans l'espace de noms qui contient le module. Pour plus d'informations, consultez Promotion de type.
- **Qualification.** Un projet peut contenir plusieurs modules, et vous pouvez déclarer des membres avec le même nom dans au moins deux modules. Toutefois, vous devez qualifier toute référence à ce membre avec le nom de module approprié si la référence est externe à ce module. Pour plus d'informations, consultez Résolution d'une référence lorsque plusieurs variables ont le même nom.

Exemple
VB

```
Public Module thisModule
    Sub Main()
        Dim userName As String = InputBox("What is your name?")
        MsgBox("User name is" & userName)
    End Sub
    ' Insert variable, property, procedure, and event declarations.
End Module
```

Voir aussi
Référence
Class, instruction (Visual Basic)
Namespace, instruction
Structure, instruction
Interface, instruction (Visual Basic)
Property, instruction
Concepts
Promotion de type

Namespace, instruction

Déclare le nom d'un espace de noms et provoque la compilation du code source qui suit la déclaration dans cet espace de noms.

```
Namespace { name | name.name }
    [ componenttypes ]
End Namespace
```

Éléments

name

Obligatoire. Nom univoque d'identification de l'espace de noms. Doit être un identificateur Visual Basic valide. Pour plus d'informations, consultez Noms d'éléments déclarés.

componenttypes

Facultatif. Éléments qui composent l'espace de noms. Ceux-ci incluent (sans s'y limiter) les énumérations, structures, interfaces, classes, modules, délégués et d'autres espaces de noms.

End Namespace

Met fin à un bloc **Namespace**.

Notes

Les espaces de noms sont utilisés comme un système d'organisation. Ils permettent de classer et de présenter des éléments de programmation exposés à d'autres programmes et applications. Notez qu'un espace de noms n'est pas un *type* dans le sens où une classe ou une structure l'est ; vous ne pouvez pas déclarer un élément de programmation comme ayant le type de données d'un espace de noms.

Vous pouvez utiliser une instruction **Namespace** uniquement au niveau du fichier ou de l'espace de noms. Cela signifie que le *contexte de déclaration* pour un espace de noms doit être un fichier source ou un autre espace de noms, et ne peut pas être une classe, une structure, un module, une interface ou une procédure. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les espaces de noms sont traités comme s'ils disposent d'un niveau d'accès **Public**. Un espace de noms peut être accessible à partir du code depuis n'importe quel emplacement dans le même projet, dans d'autres projets qui référencent le projet et dans un assembly généré à partir du projet.

Règles

- **Attributs.** Vous ne pouvez pas appliquer d'attributs à un espace de noms. Un attribut fournit des informations aux métadonnées de l'assembly, qui ne sont pas explicites pour les classifieurs source tels que les espaces de noms.
- **Modificateurs.** Vous ne pouvez pas appliquer de modificateurs d'accès ou de procédure, ou d'autres modificateurs, à un espace de noms. L'espace de noms n'étant pas un type, ces modificateurs ne sont pas explicites.
- **Imbrication.** Vous pouvez déclarer un espace de noms dans un autre. Aucune limite stricte n'est appliquée aux niveaux d'imbrication que vous pouvez déclarer, mais n'oubliez pas qu'e tout autre code qui accède aux éléments déclarés dans l'espace de noms le plus profond doit utiliser une chaîne de qualification qui contient tout les noms d'espace de noms dans la hiérarchie d'imbrication.

Comportement

- **Niveau d'accès.** Les éléments de programmation déclarés au niveau de l'espace de noms, c'est-à-dire dans un espace de noms mais pas à l'intérieur d'un autre élément, peuvent disposer d'un accès **Public** ou **Friend**. S'il n'est pas spécifié, le niveau d'accès de cet élément prend par défaut la valeur **Friend**. Les éléments que vous pouvez déclarer au niveau de l'espace de noms sont les classes, structures, modules, interfaces, énumérations et délégués. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.
- **Compilation.** Tous les éléments de programmation déclarés après une instruction **Namespace** appartiennent à cet espace de noms. Visual Basic continue à compiler les éléments dans le dernier espace de noms déclaré jusqu'à ce qu'il rencontre une instruction **End Namespace** ou une autre instruction **Namespace**.

- **Ajout à un espace de noms.** Si un espace de noms est déjà défini, même à l'extérieur de votre projet, vous pouvez y ajouter des éléments de programmation. Pour ce faire, vous utilisez une instruction **Namespace** pour demander à Visual Basic de compiler les éléments suivants dans cet espace de noms.

  L'exemple suivant définit la structure d'une nouvelle classe de liste générique et l'ajoute à l'espace de noms System.Collections.Generic.

  ```
  Namespace System.Collections.Generic
      Class specialSortedList(Of T)
          Inherits List(Of T)
          ' Insert code to define the special generic list class.
      End Class
  End Namespace
  ```

- **Espace de noms racine.** Tous les espaces de noms contenus dans votre projet sont basés sur un *espace de noms racine*. Visual Studio assigne votre nom de projet en tant qu'espace de noms racine par défaut pour l'ensemble du code de votre projet. Par exemple, si votre projet est nommé `Payroll`, ses éléments de programmation appartiennent à l'espace de noms `Payroll`. Si vous déclarez `Namespace funding`, le nom complet de cet espace de noms est `Payroll.funding`.

  Si vous souhaitez spécifier un espace de noms existant dans une instruction **Namespace**, comme dans l'exemple de classe de liste générique, vous pouvez affecter une valeur null à votre espace de noms racine. Pour ce faire, sélectionnez **Propriétés du projet** dans le menu **Projet**, puis supprimez l'entrée **Espace de noms racine** afin que la zone soit vide. Si vous ne le faites pas dans l'exemple de classe de liste générique, le compilateur Visual Basic prend `System.Collections.Generic` comme nouvel espace de noms dans le projet `Payroll`, avec le nom complet `Payroll.System.Collections.Generic`.

  Vous pouvez également utiliser le mot clé Global pour faire référence aux éléments des espaces de noms définis à l'extérieur de votre projet. Cela vous permet de conserver votre nom de projet comme espace de noms racine. Cela réduit également le risque de fusion involontaire de vos éléments de programmation avec ceux des espaces de noms existants.

Dépannage

- **Types non définis.** L'espace de noms racine peut provoquer des concaténations inattendues des noms d'espace de noms. Si vous créez une référence aux espaces de noms définis à l'extérieur de votre projet, le compilateur Visual Basic peut les interpréter comme des espaces de noms imbriqués dans votre espace de noms racine. Dans ce cas, le compilateur ne reconnaît pas les types qui ont déjà été définis dans les espaces de noms externes. Pour éviter cela, affectez une valeur null à votre espace de noms racine, comme décrit dans « Espace de noms racine », ou utilisez le mot clé **Global** pour accéder aux éléments des espaces de noms externes.

Exemple

L'exemple suivant déclare deux espaces de noms, l'un imbriqué dans l'autre.

VB

```
Namespace n1
    Namespace n2
        Class a
            ' Insert class definition.
        End Class
    End Namespace
End Namespace
```

L'exemple suivant déclare plusieurs espaces de noms imbriqués sur une seule ligne, et est équivalent à l'exemple précédent.

VB

```
Namespace n1.n2
    Class a
        ' Insert class definition.
    End Class
End Namespace
```

L'exemple suivant accède à la classe définie dans les précédents exemples.

VB

```
Dim instance As New n1.n2.a
```

Voir aussi

Référence

Imports, instruction

Global

Concepts

Noms d'éléments déclarés

Espaces de noms dans Visual Basic

On Error, instruction (Visual Basic)

Active une routine de gestion des erreurs et spécifie l'emplacement de la routine dans une procédure ; peut également être utilisée pour désactiver une routine de gestion des erreurs.

Sans instruction **On Error**, une erreur d'exécution est irrécupérable : un message d'erreur s'affiche et l'exécution s'arrête.

Nous vous suggérons, dans la mesure du possible, d'utiliser une gestion structurée des exceptions dans votre code plutôt qu'une gestion non structurée et l'instruction **On Error**. Pour plus d'informations, consultez Gestion structurée des exceptions en Visual Basic.

```
On Error { GoTo [ line | 0 | -1 ] | Resume Next }
```

Éléments

GoTo *line*

Active la routine de gestion des erreurs qui démarre à la ligne spécifiée dans l'argument *line* requis. L'argument *line* est une étiquette de ligne ou un numéro de ligne. Si une erreur d'exécution se produit, le contrôle effectue un branchement vers la ligne spécifiée, ce qui active le gestionnaire d'erreurs. La ligne spécifiée doit se trouver dans la même procédure que l'instruction **On Error** ; sinon, une erreur de compilation se produit.

GoTo 0

Désactive le gestionnaire d'erreurs activé dans la procédure en cours et le réinitialise à la valeur **Nothing**.

GoTo -1

Désactive l'exception activée dans la procédure en cours et la réinitialise à la valeur **Nothing**.

Resume Next

Indique que dans le cadre d'une erreur d'exécution, le contrôle passe à l'instruction qui suit directement celle où s'est produite l'erreur ; l'exécution se poursuit à partir de ce point. Utilisez ce formulaire plutôt que **On Error GoTo** lors de l'accès aux objets.

Notes

Un gestionnaire d'erreurs « actif » est activé par une instruction **On Error**. Un gestionnaire d'erreurs « actif » est un gestionnaire activé qui figure dans le processus de gestion d'une erreur.

Si une erreur se produit lorsqu'un gestionnaire d'erreurs est actif (entre l'occurrence de l'erreur et l'instruction **Resume**, **Exit Sub**, **Exit Function** ou **Exit Property**), le gestionnaire d'erreurs de la procédure en cours ne peut pas gérer l'erreur. Le contrôle retourne à la procédure appelante.

Si la procédure appelante possède un gestionnaire d'erreurs activé, elle peut gérer l'erreur. Si le gestionnaire d'erreurs de la procédure appelante est également actif, le contrôle est retourné par l'intermédiaire des procédures appelantes précédentes jusqu'à la découverte d'un gestionnaire d'erreurs activé (mais inactif). Si aucun gestionnaire d'erreurs de ce type n'est découvert, l'erreur est irrécupérable au point où elle est apparue.

Chaque fois que le gestionnaire d'erreurs retourne le contrôle à une procédure appelante, cette procédure devient la procédure en cours. Dès qu'une erreur est gérée par une gestionnaire d'erreurs dans une procédure, l'exécution reprend dans la procédure en cours au point désigné par l'instruction **Resume**.

Remarque
Une routine de gestion d'erreurs n'est pas une procédure **Sub** ni une procédure **Function**. C'est une section de code marqué e par une étiquette de ligne ou un numéro de ligne.

Number, propriété

Les routines de gestion d'erreurs reposent sur la valeur de la propriété **Number** de l'objet **Err** pour déterminer la cause de l'erreur. La routine devrait tester ou enregistrer les valeurs de propriétés adéquates dans l'objet **Err** avant qu'une autre erreur ne se produise ou avant qu'une procédure pouvant engendrer une erreur soit appelée. Les valeurs de propriété de l'objet **Err** reflètent uniquement l'erreur la plus récente. Le message d'erreur associé à **Err.Number** est contenu dans **Err.Description**.

Throw, instruction

Une erreur qui est déclenchée avec la méthode **Err.Raise** affecte à la propriété **Exception** une instance nouvellement créée de la classe Exception. Afin de prendre en charge le déclenchement d'exceptions des types d'exception dérivés, une instruction **Throw** est prise en charge dans le langage. Elle prend un seul paramètre qui est une instance d'exception à lever. L'exemple suivant montre comment ces fonctionnalités peuvent être utilisées avec la prise en charge de gestion des exceptions existante :

VB

```
    On Error GoTo Handler
    Throw New DivideByZeroException()
Handler:
    If (TypeOf Err.GetException() Is DivideByZeroException) Then
    ' Code for handling the error is entered here.
    End If
```

Notez que l'instruction **On Error GoTo** intercepte toutes les erreurs quelle que soit la classe d'exception.

On Error Resume Next

On Error Resume Next poursuit l'exécution par l'instruction qui suit directement celle qui a engendré l'erreur d'exécution ou par celle qui suit directement le dernier appel de la procédure contenant l'instruction **On Error Resume Next**. Cette instruction permet la poursuite de l'exécution malgré l'erreur d'exécution. Vous pouvez placer la routine de gestion des erreurs à l'endroit où l'erreur risque de se produire au lieu de transférer le contrôle à un autre emplacement de la procédure. Une instruction **On Error Resume Next** devient inactive lors de l'appel d'une autre procédure ; vous devriez donc exécuter une instruction **On Error Resume Next** dans chaque routine appelée si vous souhaitez utiliser une gestion des erreurs inline dans cette routine.

Remarque

La construction **On Error Resume Next** peut être préférable à **On Error GoTo** lors de la gestion des erreurs générées au co urs de l'accès à d'autres objets. La vérification de **Err** après chaque interaction avec un objet supprime l'ambiguïté selon laqu elle l'objet est accessible par le code. Vous pouvez déterminer avec certitude l'objet ayant déclenché le code d'erreur dans **Err .Number**, ainsi que l'objet qui a généré l'erreur à l'origine (l'objet spécifié dans **Err.Source**).

On Error GoTo 0

On Error GoTo 0 désactive la gestion des erreurs dans la procédure actuelle. Il n'indique pas que la ligne 0 marque le début du code de gestion des erreurs, même si la procédure contient une ligne numérotée 0. En l'absence d'une instruction **On Error GoTo 0**, un gestionnaire d'erreurs est automatiquement désactivé lorsqu'une procédure est quittée.

On Error GoTo -1

On Error GoTo -1 désactive l'exception dans la procédure actuelle. Il n'indique pas que la ligne -1 marque le début du code de gestion des erreurs, même si la procédure contient une ligne numérotée -1. En l'absence d'une instruction **On Error GoTo -1**, une exception est automatiquement désactivée lorsqu'une procédure est quittée.

Pour empêcher l'exécution du code de gestion des erreurs lorsque aucune erreur ne se produit, insérez une instruction **Exit Sub**, **Exit Function** ou **Exit Property** immédiatement avant la routine de gestion des erreurs, comme illustré dans le fragment suivant :

VB

```
Public Sub InitializeMatrix(ByVal Var1 As Object, ByVal Var2 As Object)
   On Error GoTo ErrorHandler
   ' Insert code that might generate an error here
   Exit Sub
ErrorHandler:
   ' Insert code to handle the error here
   Resume Next
End Sub
```

Dans cet exemple, le code de gestion des erreurs suit l'instruction **Exit Sub** et précède l'instruction **End Sub** pour la séparer du flux de procédure. Vous pouvez placer du code de gestion des erreurs à n'importe quel endroit d'une procédure.

Erreurs non interceptées

Les erreurs non interceptées dans les objets sont retournées à l'application de contrôle lorsque l'objet s'exécute comme un fichier exécutable. Dans l'environnement de développement, les erreurs non interceptées sont retournées à l'application de

contrôle uniquement si les options correctes sont définies. Consultez la documentation de votre application hôte pour obtenir une description des options à définir lors du débogage et savoir comment les définir, ainsi que pour déterminer si l'hôte peut créer des classes.

Si vous créez un objet qui accède à d'autres objets, vous devez essayer de traiter les erreurs non gérées retournées. Dans le cas contraire, vous pouvez mapper les codes d'erreur de **Err.Number** vers l'une de vos erreurs, puis passer ces erreurs à l'appelant de votre objet. Vous devez spécifier votre erreur en ajoutant votre code d'erreur à la constante **VbObjectError**. Par exemple, si votre code d'erreur est 1052, vous devez l'assigner comme suit :

VB

```
Err.Number = vbObjectError + 1052
```

Attention

Les erreurs système lors des appels à des bibliothèques de liens dynamiques (DLL, Dynamic-Link Libraries) Windows ne déclenchent pas d'exceptions et ne peuvent pas être décelées par l'interception des erreurs Visual Basic. Lors d'un appel aux fonctions DLL, vous devez vérifier si chaque valeur de retour a abouti ou échoué (selon les spécifications API) ; en cas d'échec, vérifiez la valeur de la propriété **LastDLLError** dans l'objet **Err**.

Exemple

Cet exemple utilise d'abord l'instruction **On Error GoTo** pour spécifier l'emplacement d'une routine de gestion des erreurs dans une procédure. Dans l'exemple, la tentative de division par zéro génère une erreur numéro 6. L'erreur est gérée dans la routine de gestion des erreurs, et le contrôle est ensuite retourné à l'instruction qui a engendré l'erreur. L'instruction **On Error GoTo 0** désactive l'interception des erreurs. L'instruction **On Error Resume Next** est ensuite utilisée pour différer l'interception des erreurs de sorte que le contexte de l'erreur générée par l'instruction suivante peut être déterminé avec certitude. Notez que **Err.Clear** est utilisé pour effacer les propriétés de l'objet **Err** après la gestion de l'erreur.

VB

```
Public Sub OnErrorDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Dim x As Integer = 32
   Dim y As Integer = 0
   Dim z As Integer
   z = x / y   ' Creates a divide by zero error
   On Error GoTo 0   ' Turn off error trapping.
   On Error Resume Next   ' Defer error trapping.
   z = x / y   ' Creates a divide by zero error again
   If Err.Number = 6 Then
      ' Tell user what happened. Then clear the Err object.
      Dim Msg As String
      Msg = "There was an error attempting to divide by zero!"
      MsgBox(Msg, , "Divide by zero error")
      Err.Clear() ' Clear Err object fields.
   End If
Exit Sub      ' Exit to avoid handler.
ErrorHandler:  ' Error-handling routine.
   Select Case Err.Number   ' Evaluate error number.
      Case 6   ' Divide by zero error
         MsgBox("You attempted to divide by zero!")
         ' Insert code to handle this error
      Case Else
         ' Insert code to handle other situations here...
   End Select
   Resume Next  ' Resume execution at same line
                ' that caused the error.
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic

Référence

End, instruction

Err, objet (Visual Basic)

Exit, instruction (Visual Basic)

LastDllError, propriété (objet Err)

Resume, instruction

Operator, instruction

Déclare le symbole d'opérateur, les opérandes et le code qui définissent une procédure d'opérateur sur une classe ou une structure.

```
[ <attrlist> ] Public [ Overloads ] Shared [ Shadows ] [ Widening | Narrowing ]
Operator operatorsymbol ( operand1 [, operand2 ]) [ As [ <attrlist> ] type ]
    [ statements ]
    [ statements ]
    Return returnvalue
    [ statements ]
End Operator
```

Éléments

attrlist

Facultatif. Consultez Liste d'attributs.

Public

Obligatoire. Indique que cette procédure d'opérateur dispose d'un accès Public (Visual Basic).

Overloads

Facultatif. Consultez Overloads.

Shared

Obligatoire. Indique que cette procédure d'opérateur est une procédure Shared (Visual Basic).

Shadows

Facultatif. Consultez Shadows.

Widening

Obligatoire pour un opérateur de conversion sauf si vous spécifiez **Narrowing**. Indique que cette procédure d'opérateur définit une conversion Widening. Consultez « Conversions étendues et restrictives » sur cette page d'aide.

Narrowing

Obligatoire pour un opérateur de conversion sauf si vous spécifiez **Widening**. Indique que cette procédure d'opérateur définit une conversion Narrowing. Consultez « Conversions étendues et restrictives » sur cette page d'aide.

operatorsymbol

Obligatoire. Symbole ou identificateur de l'opérateur défini par cette procédure d'opérateur.

operand1

Obligatoire. Nom et type de l'opérande unique d'un opérateur unaire (y compris un opérateur de conversion) ou opérande gauche d'un opérateur binaire.

operand2

Obligatoire pour les opérateurs binaires. Nom et type de l'opérande droit d'un opérateur binaire.

operand1 et *operand2* emploient la syntaxe et les paramètres suivants :

```
[ ByVal ] operandname [ As operandtype ]
```

Élément	Description
ByVal	Facultatif, mais le mécanisme de passage doit être ByVal.
operandname	Obligatoire. Nom de la variable représentant cet opérande. Consultez Noms d'éléments déclarés.

operandtype	Facultatif sauf si **Option Strict** a la valeur **On**. Type de données de cet opérande.

type

Facultatif sauf si **Option Strict** a la valeur **On**. Type de données de la valeur retournée par la procédure d'opérateur.

statements

Facultatif. Bloc d'instructions exécutées par la procédure d'opérateur.

returnvalue

Obligatoire. Valeur retournée par la procédure d'opérateur au code appelant.

End Operator

Obligatoire. Met fin à la définition de cette procédure d'opérateur.

Notes

Vous pouvez utiliser **Operator** uniquement dans une classe ou une structure. Cela signifie que le *contexte de déclaration* pour un opérateur ne peut pas être un fichier source, un espace de noms, un module, une interface, une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Tous les opérateurs doivent être **Public Shared**. Vous ne pouvez pas spécifier **ByRef**, **Optional** ou **ParamArray** pour l'un ou l'autre des opérandes.

Vous ne pouvez pas utiliser le symbole ou l'identificateur d'opérateur pour stocker une valeur de retour. Vous devez utiliser l'instruction **Return**, et elle doit spécifier une valeur. Il n'existe pas de limite au nombre d'instructions **Return** pouvant apparaître n'importe où dans la procédure.

Cette définition d'un opérateur est appelée *surcharge d'opérateur*, que vous utilisiez ou non le mot clé **Overloads**. Le tableau suivant présente les opérateurs que vous pouvez définir.

Type	**Opérateurs**
Unaire	**+**, **-**, **IsFalse**, **IsTrue**, **Not**
Binaire	**+**, **-**, *****, **/**, ****, **&**, **^**, **>>**, **<<**, **=**, **<>**, **>**, **>=**, **<**, **<=**, **And**, **Like**, **Mod**, **Or**, **Xor**
Conversion (unaire)	**CType**

Notez que l'opérateur **=** dans la liste binaire est l'opérateur de comparaison, et non l'opérateur d'assignation.

Lorsque vous définissez **CType**, vous devez spécifier **Widening** ou **Narrowing**.

Paires appariées

Vous devez définir certains opérateurs comme des paires appariées. Si vous définissez l'un des opérateurs de cette paire, vous devez également définir l'autre. Les paires appariées sont les suivantes :

- **=** et **<>**
- **>** et **<**
- **>=** et **<=**
- **IsTrue** et **IsFalse**

Restrictions de type de données

Chaque opérateur que vous définissez doit impliquer la classe ou la structure sur laquelle vous le définissez. Cela signifie que la classe ou la structure doit être définie comme le type de données des éléments suivants :

- l'opérande d'un opérateur unaire ;
- au moins l'un des opérandes d'un opérateur binaire ;
- l'opérande ou le type de retour d'un opérateur de conversion.

Certains opérateurs présentent d'autres restrictions de type de données, à savoir :

- Si vous définissez les opérateurs **IsTrue** et **IsFalse**, tous deux doivent retourner le type **Boolean**.
- Si vous définissez les opérateurs **<<** et **>>**, tous deux doivent spécifier le type **Integer** pour *operandtype* de *operand2*.

Le type de retour ne doit pas correspondre au type de l'un ou l'autre des opérandes. Par exemple, un opérateur de comparaison tel que **=** ou **<>** peut retourner **Boolean** même si aucun des opérandes n'a la valeur **Boolean**.

Opérateurs logiques et au niveau du bit

Les opérateurs **And**, **Or**, **Not** et **Xor** peuvent exécuter des opérations logiques ou au niveau du bit en Visual Basic. Toutefois, si vous définissez l'un de ces opérateurs sur une classe ou une structure, vous pouvez définir uniquement son opération au niveau du bit.

Vous ne pouvez pas définir directement l'opérateur **AndAlso** avec une instruction **Operator**. Toutefois, vous pouvez utiliser **AndAlso** si vous avez rempli les conditions suivantes :

- Vous avez défini **And** sur les mêmes types d'opérande que vous souhaitez utiliser pour **AndAlso**.
- Votre définition de **And** retourne le même type que la classe ou la structure sur laquelle vous l'avez défini.
- Vous avez défini l'opérateur **IsFalse** sur la classe ou la structure sur laquelle vous avez défini **And**.

De la même façon, vous pouvez utiliser **OrElse** si vous avez défini **Or** sur les mêmes opérandes, avec le type de retour de la classe ou la structure, et si vous avez défini **IsTrue** sur la classe ou la structure.

Conversions étendues et restrictives

Une *conversion étendue* réussit toujours au moment de l'exécution, alors qu'une *conversion restrictive* peut échouer au moment de l'exécution. Pour plus d'informations, consultez Conversions étendues et restrictives.

Si vous déclarez une procédure de conversion comme étant **Widening**, votre code de procédure ne doit pas générer d'échecs. Notamment :

- Elle doit toujours retourner une valeur valide de type *type*.
- Elle doit gérer toutes les exceptions possibles et les autres conditions d'erreur.
- Elle doit gérer les retours en cas d'erreur des procédures qu'elle appelle.

S'il existe un risque d'échec d'une procédure de conversion, ou si celle-ci peut générer une exception non gérée, vous devez la déclarer comme étant **Narrowing**.

Exemple

L'exemple de code suivant utilise l'instruction **Operator** pour définir le plan d'une structure qui contient des procédures d'opérateur pour les opérateurs **And**, **Or**, **IsFalse** et **IsTrue**. Les opérateurs **And** et **Or** acceptent chacun deux opérandes de type `abc` et de type de retour `abc`. Les opérateurs **IsFalse** et **IsTrue** acceptent chacun un opérande unique de type `abc` et de retour **Boolean**. Ces définitions permettent au code appelant d'utiliser **And**, **AndAlso**, **Or** et **OrElse** avec des opérandes de type `abc`.

VB

```
Public Structure abc
    Dim d As Date
    Public Shared Operator And(ByVal x As abc, ByVal y As abc) As abc
        Dim r As New abc
        ' Insert code to calculate And of x and y.
        Return r
    End Operator
    Public Shared Operator Or(ByVal x As abc, ByVal y As abc) As abc
        Dim r As New abc
        ' Insert code to calculate Or of x and y.
        Return r
    End Operator
    Public Shared Operator IsFalse(ByVal z As abc) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsFalse of z.
        Return b
    End Operator
    Public Shared Operator IsTrue(ByVal z As abc) As Boolean
        Dim b As Boolean
```

```
        ' Insert code to calculate IsTrue of z.
        Return b
    End Operator
End Structure
```

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Comment : appeler une procédure d'opérateur

Comment : utiliser une classe qui définit des opérateurs

Référence

IsFalse, opérateur

IsTrue, opérateur

Widening

Narrowing

Concepts

Conversions étendues et restrictives

Procédures d'opérateur

Option Compare, instruction

Déclare la méthode de comparaison par défaut à utiliser lors de la comparaison de données de chaîne.

```
Option Compare { Binary | Text }
```

Éléments

Binary

Facultatif. Donne lieu à des comparaisons de chaînes basées sur un ordre de tri dérivé de la représentation binaire interne des caractères.

Text

Facultatif. Fournit des comparaisons de chaînes basées sur un ordre de tri ne respectant pas la casse et qui est déterminé par les paramètres régionaux de votre système.

Notes

Si elle est utilisée, l'instruction **Option Compare** doit apparaître dans un fichier avant les autres instructions du code source.

L'instruction **Option Compare** spécifie la méthode de comparaison de chaînes (**Binary** ou **Text**) pour une classe, un module ou une structure. Si le module ne contient pas d'instruction **Option Compare**, la méthode de comparaison de texte par défaut est de type **Binary**.

Dans Microsoft Windows, l'ordre de tri est déterminé par la page de codes. Pour plus d'informations, consultez Code Pages.

Dans l'exemple suivant, les caractères de la page de codes Anglais/Européen (ANSI 1252) sont triés à l'aide de **Option Compare Binary**, qui génère un ordre de tri binaire standard.

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

Lorsque les mêmes caractères de la même page de codes sont triés à l'aide de **Option Compare Text**, on obtient l'ordre de tri de texte ci-dessous.

```
(A=a) < (À = à) < (B=b) < (E=e) < (Ê = ê) < (Z=z) < (Ø = ø)
```

Vous pouvez également définir **Option Compare** dans l'environnement de développement intégré (IDE) de Visual Studio ou sur une ligne de commande.

Pour définir Option Compare dans l'environnement de développement intégré (IDE)

1. Dans le menu **Outils**, choisissez **Options**.
2. Ouvrez le nœud **Projets et solutions**.
3. Choisissez **Valeurs par défaut VB**.
4. Modifiez le paramètre **Option Compare**.

Pour définir Option Compare sur la ligne de commande

- Incluez l'option de compilateur /optioncompare dans la commande **vbc**.

Exemple

L'exemple suivant utilise l'instruction **Option Compare** pour définir la comparaison binaire comme méthode de comparaison de chaînes par défaut.

VB

```
' Set the string comparison method to Binary ("AAA" < "aaa").
Option Compare Binary
```

L'exemple suivant utilise l'instruction **Option Compare** pour définir l'ordre de tri de texte sans respect de la casse comme méthode de comparaison de chaînes par défaut.

VB

```
' Set the string comparison method to Text ("AAA" = "aaa").
Option Compare Text
```

Voir aussi

Référence

Opérateurs de comparaison (Visual Basic)
InStr, fonction (Visual Basic)
InStrRev, fonction (Visual Basic)
Replace, fonction (Visual Basic)
Split, fonction (Visual Basic)
StrComp, fonction (Visual Basic)
/optioncompare
Option Explicit, instruction (Visual Basic)
Option Strict, instruction
/optionexplicit
/optionstrict
Valeurs par défaut VB, Projets, boîte de dialogue Options

Option Explicit, instruction (Visual Basic)

Force une déclaration explicite de toutes les variables dans un fichier.

```
Option Explicit { On | Off }
```

Éléments

On

Facultatif. Active la vérification **Option Explicit**. Si **On** ou **Off** n'est pas spécifié, la valeur par défaut est **On**.

Off

Facultatif. Désactive la vérification **Option Explicit**.

Notes

Si elle est utilisée, l'instruction **Option Explicit** doit apparaître dans un fichier avant toute autre instruction de code source.

Si l'instruction **Option Explicit** apparaît dans un fichier, vous devez déclarer explicitement toutes les variables à l'aide des instructions **Dim** ou **ReDim**. Si vous tentez d'utiliser un nom de variable non déclarée, une erreur se produit au moment de la compilation.

Utilisez l'instruction **Option Explicit** pour éviter toute faute de frappe lors de la saisie du nom d'une variable existante ou pour écarter tout risque de confusion dans un code où la portée de la variable n'apparaît pas clairement. Si vous n'utilisez pas l'instruction **Option Explicit**, toutes les variables non déclarées prennent le type **Object**.

Remarque
Si vous ne spécifiez pas **Option Explicit** dans votre code, le compilateur par défaut est **Option Explicit On**.

Vous pouvez également définir **Option Explicit** dans l'environnement de développement intégré (IDE) Visual Studio ou dans une ligne de commande.

Pour définir Option Explicit dans l'environnement de développement intégré (IDE)

1. Dans le menu **Outils**, choisissez **Options**.
2. Ouvrez le nœud **Projets et Solutions**.
3. Choisissez **Valeurs par défaut VB**.
4. Modifiez la propriété **Option Explicit**.

Pour définir Option Explicit dans la ligne de commande

- Incluez l'option du compilateur /optionexplicit dans la commande **vbc**.

Exemple

L'exemple suivant utilise l'instruction **Option Explicit** pour forcer la déclaration explicite de toutes les variables. Toute tentative d'utilisation d'une variable non déclarée se traduit par une erreur au moment de la compilation.

VB

```
' Force explicit variable declaration.
Option Explicit On
```

VB

```
Dim thisVar As Integer
thisVar = 10
' The following assignment produces a COMPILER ERROR because
```

```
' the variable is not declared and Option Explicit is On.
thisInt = 10 ' causes ERROR
```

Voir aussi

Référence

Dim, instruction (Visual Basic)

ReDim, instruction (Visual Basic)

Option Compare, instruction

Option Strict, instruction

/optioncompare

/optionexplicit

/optionstrict

Valeurs par défaut VB, Projets, boîte de dialogue Options

Option Strict, instruction

Limite les conversions de types de données implicites aux seules conversions *étendues*.

```
Option Strict { On | Off }
```

Éléments

On

Facultatif. Active le contrôle **Option Strict**.

Off

Facultatif. Désactive le contrôle **Option Strict**. Si **On** ou **Off** n'est pas spécifié, la valeur par défaut est **Off**.

Notes

Si elle est utilisée, l'instruction **Option Strict** doit apparaître dans un fichier avant les autres instructions du code source.

Visual Basic autorise les conversions de plusieurs types de données en d'autres types de données. Une perte de données peut se produire si la valeur de l'un des types de données est convertie en un type de données dont la précision ou la capacité est moindre. Une erreur d'exécution se produit en cas d'échec d'une telle *conversion restrictive*. **Option Strict** veille à ce que ces conversions restrictives soient notifiées au moment de la compilation afin qu'elles puissent être évitées.

En plus d'interdire des conversions restrictives implicites, **Option Strict** génère une erreur pour la *liaison tardive*. Un objet est à liaison tardive lorsqu'il est assigné à une variable déclarée comme étant du type **Object**.

Étant donné que **Option Strict On** fournit un *typage fort*, empêche les conversions de type involontaires susceptibles d'entraîner une perte de données, interdit les liaisons tardives et améliore les performances, il est fortement recommandé de l'utiliser.

Remarque
La valeur par défaut du compilateur est **Option Strict Off** si vous ne spécifiez pas **Option Strict** dans votre code.

Vous pouvez également définir **Option Strict** dans l'environnement de développement intégré (IDE) de Visual Studio ou sur une ligne de commande.

Pour définir Option Strict dans l'environnement de développement intégré (IDE)

1. Dans le menu **Outils**, choisissez **Options**.
2. Ouvrez le nœud **Projets et solutions**.
3. Choisissez **Valeurs par défaut VB**.
4. Modifiez le paramètre **Option Strict**.

Pour définir Option Strict sur la ligne de commande

- Incluez l'option de compilateur /optionstrict dans la commande **vbc**.

Exemple

L'exemple suivant montre comment l'instruction **Option Strict** interdit les liaisons tardives et les conversions où des données pourraient se perdre.

VB

```
Option Strict On
```

VB

```
Dim thisVar As Integer
Dim thisObj As Object = New widget
thisVar = 1000   ' Declared variable does not generate error.
' Attempting to convert Double to Integer generates a COMPILER ERROR.
thisVar = 1234567890.9876542 ' causes ERROR
' Late-bound call generates a COMPILER ERROR.
Call thisObj.Method1() ' causes ERROR
```

Voir aussi

Référence

Option Compare, instruction

Option Explicit, instruction (Visual Basic)

/optioncompare

/optionexplicit

/optionstrict

Valeurs par défaut VB, Projets, boîte de dialogue Options

Concepts

Conversions étendues et restrictives

Property, instruction

Déclare le nom d'une propriété, ainsi que les procédures de propriété utilisées pour stocker et récupérer la valeur de la propriété.

```
[ <attributelist> ] [ Default ] [ accessmodifier ]
[ propertymodifiers ] [ Shared ] [ Shadows ] [ ReadOnly | WriteOnly ]
Property name ( [ parameterlist ] ) [ As returntype ] [ Implements implementslist ]
    [ <attributelist> ] [ accessmodifier ] Get
        [ statements ]
    End Get
    [ <attributelist> ] [ accessmodifier ] Set ( ByVal value As returntype [, parameterlist
 ] )
        [ statements ]
    End Set
End Property
```

Éléments

attributelist

Facultatif. Liste des attributs s'appliquant à cette propriété ou à la procédure **Get** ou **Set**. Consultez Liste d'attributs.

Default

Facultatif. Spécifie que cette propriété est la propriété par défaut pour la classe ou la structure sur laquelle elle est définie. Les propriétés par défaut doivent accepter des paramètres et peuvent être définies et récupérées sans la spécification du nom de la propriété. Si vous déclarez la propriété comme **Default**, vous ne pouvez pas utiliser **Private** sur la propriété ou sur l'une ou l'autre de ses procédures de propriété.

accessmodifier

Facultatif sur l'instruction **Property** et sur, au plus, l'une des instructions **Get** et **Set**. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

propertymodifiers

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Overloads
- Overrides
- Overridable
- NotOverridable
- MustOverride
- **MustOverride Overrides**
- **NotOverridable Overrides**

Shared

Facultatif. Consultez Shared (Visual Basic).

Shadows

Facultatif. Consultez Shadows.

ReadOnly

Facultatif. Consultez ReadOnly (Visual Basic).

WriteOnly

Facultatif. Consultez WriteOnly.

name

Obligatoire. Nom de la propriété. Consultez Noms d'éléments déclarés.

parameterlist

Facultatif. Liste des noms de variables locales qui représentent les paramètres de cette propriété, et les éventuels paramètres supplémentaires de la procédure **Set**. Consultez Liste de paramètres.

returntype

Obligatoire si **Option Strict** a la valeur **On**. Type de données de la valeur retournée par cette propriété.

Implements

Facultatif. Indique que cette propriété implémente une ou plusieurs propriétés, dont chacune est définie dans une interface implémentée par la classe ou la structure conteneur de cette propriété. Consultez Implements, instruction.

implementslist

Requis si **Implements** est fourni. Liste des propriétés implémentées.

```
implementedproperty [ , implementedproperty ... ]
```

Chaque *implementedproperty* emploie la syntaxe et les paramètres suivants :

```
interface.definedname
```

Élément	Description
interface	Obligatoire. Nom d'une interface implémentée par la classe ou la structure conteneur de cette propriété.
definedname	Obligatoire. Nom par lequel la propriété est définie dans *interface*.

Get

Obligatoire à moins que la propriété soit marquée **WriteOnly**. Lance une procédure de propriété **Get** utilisée pour retourner la valeur de la propriété.

statements

Facultatif. Bloc d'instructions à exécuter dans la procédure **Get** ou **Set**.

End Get

Met fin à la procédure de propriété **Get**.

Set

Obligatoire à moins que la propriété soit marquée **ReadOnly**. Lance une procédure de propriété **Set** utilisée pour enregistrer la valeur de la propriété.

End Set

Met fin à la procédure de propriété **Set**.

End Property

Met fin à la définition de cette propriété.

Notes

L'instruction **Property** présente la déclaration d'une propriété. Une propriété peut contenir une procédure **Get** (lecture seule), une procédure **Set** (écriture seule), ou les deux (lecture-écriture).

Vous pouvez utiliser **Property** seulement au niveau du module. Cela signifie que le *contexte de déclaration* pour une propriété doit être une classe, une structure, un module ou une interface, et ne peut pas être un fichier source, un espace de noms, une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les propriétés disposent par défaut d'un accès public. Vous pouvez modifier le niveau d'accès d'une propriété à l'aide d'un modificateur d'accès sur l'instruction **Property**, et vous pouvez éventuellement appliquer un niveau d'accès plus restrictif à l'une de ses procédures de propriété.

Visual Basic passe un paramètre à la procédure **Set** pendant les assignations de propriétés. Si vous ne précisez pas de paramètre pour **Set**, l'environnement de développement intégré (IDE) utilise un paramètre implicite appelé `value`. Ce paramètre contient la valeur à assigner à la propriété. Vous enregistrez généralement cette valeur dans une variable locale privée et la retournez à chaque appel de la procédure **Get**.

Règles

- **Niveaux d'accès mixtes.** Si vous définissez une propriété en lecture-écriture, vous pouvez éventuellement spécifier un niveau d'accès différent pour la procédure **Get** ou **Set**, mais pas pour les deux. Dans ce cas, le niveau d'accès de la procédure doit être plus restrictif que celui de la propriété. Par exemple, si la propriété est déclarée **Friend**, vous pouvez déclarer la procédure **Set Private**, mais pas **Public**.

  Si vous définissez une propriété **ReadOnly** ou **WriteOnly**, la procédure de propriété unique (**Get** ou **Set**, respectivement) représente la propriété entière. Vous ne pouvez pas déclarer un niveau d'accès différent pour cette procédure, parce que deux niveaux d'accès seront définis pour la propriété.

- **Type de retour.** L'instruction **Property** peut déclarer le type de données de la valeur qu'elle retourne. Vous pouvez spécifier un type de données ou le nom d'une énumération, d'une structure, d'une classe ou d'une interface.

  Si vous ne spécifiez pas *returntype*, la propriété retourne **Object**.

- **Implémentation.** Si cette propriété utilise le mot clé **Implements**, la classe ou la structure conteneur doit contenir une instruction **Implements** qui suit immédiatement son instruction **Class** ou **Structure**. L'instruction **Implements** doit contenir chaque interface spécifiée dans *implementslist*. Toutefois, le nom par lequel une interface définit **Property** (dans *definedname*) ne doit pas être identique au nom de cette propriété (dans *name*).

Comportement

- **Retour d'une procédure de propriété.** Lorsque la procédure **Get** ou **Set** retourne au code appelant, l'exécution se poursuit avec l'instruction qui suit celle qui l'a appelé.

  Les instructions **Exit Property** et **Return** provoquent la sortie immédiate d'une procédure de propriété. Plusieurs instructions **Exit Property** et **Return** peuvent apparaître n'importe où dans la procédure, et vous pouvez associer des instructions **Exit Property** et **Return**.

- **Valeur de retour.** Pour retourner une valeur à partir d'une procédure **Get**, vous pouvez soit assigner la valeur au nom de la propriété, soit l'inclure dans une instruction **Return**. L'exemple suivant assigne la valeur de retour au nom de la propriété `quoteForTheDay`, puis utilise l'instruction **Exit Property** pour retourner :

  VB

  ```
  Private quoteValue As String = "No quote assigned yet."
  ```

  VB

  ```
  ReadOnly Property quoteForTheDay() As String
      Get
          quoteForTheDay = quoteValue
          Exit Property
      End Get
  End Property
  ```

  Si vous utilisez **Exit Property** sans assigner une valeur à *name*, la procédure **Get** retourne la valeur par défaut pour le type de données de la propriété.

  L'instruction **Return** assigne simultanément la valeur de retour de la procédure **Get** et ferme la procédure. L'exemple

suivant fournit une illustration.

VB

```
Private quoteValue As String = "No quote assigned yet."
```

VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        Return quoteValue
    End Get
End Property
```

Exemple

L'exemple suivant déclare une propriété dans une classe.

VB

```
Class Class1
    ' Define a local variable to store the property value.
    Private propertyValue As String
    ' Define the property.
    Public Property prop1() As String
        Get
            ' The Get property procedure is called when the value
            ' of a property is retrieved.
            Return propertyValue
        End Get
        Set(ByVal value As String)
            ' The Set property procedure is called when the value
            ' of a property is modified.  The value to be assigned
            ' is passed in the argument to Set.
            propertyValue = value
        End Set
    End Property
End Class
```

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Référence

Get, instruction

Set, instruction (Visual Basic)

Liste de paramètres

Concepts

Propriétés par défaut

Instructions Q-Z

Le tableau suivant énumère les instructions du langage Visual Basic.

RaiseEvent	ReDim	REM	RemoveHandler
Resume	Return	Select...Case	Set
Stop	Structure	Sub	SyncLock
Throw	Try...Catch...Finally	Using	While...End While
With...End With			

Voir aussi

Référence

Instructions A-E

Instructions F-P

Autres ressources

Référence Visual Basic

RaiseEvent, instruction

Déclenche un événement déclaré au niveau module dans une classe, un formulaire ou un document.

```
RaiseEvent eventname[( argumentlist )]
```

Éléments

eventname

Obligatoire. Nom de l'événement à déclencher.

argumentlist

Facultatif. Liste délimitée par des virgules énumérant des variables, des tableaux ou des expressions. L'argument *argumentlist* doit être mis entre parenthèses. En cas d'absence d'argument, les parenthèses doivent être supprimées.

Notes

Le paramètre *eventname* requis correspond au nom d'un événement déclaré dans le module. Il suit les conventions d'affectation de nom de variables Visual Basic.

Si l'événement n'a pas été déclaré dans le module dans lequel il est déclenché, une erreur se produit. Le fragment de code suivant illustre une déclaration de l'événement et une procédure dans laquelle l'événement est déclenché.

VB

```
' Declare an event at module level.
Event LogonCompleted(ByVal UserName As String)

Sub Logon(ByVal UserName As String)
    ' Raise the event.
    RaiseEvent LogonCompleted(UserName)
End Sub
```

Vous ne pouvez pas utiliser **RaiseEvent** pour générer des événements qui ne sont pas déclarés de façon explicite dans le module. Par exemple, tous les formulaires héritent d'un événement Click de System.Windows.Forms.Form ; il ne peut pas être déclenché à l'aide de **RaiseEvent** dans un formulaire dérivé. Si vous déclarez un événement `Click` dans le module formulaire, celui-ci occulte l'événement **Click** appartenant au formulaire. Vous pouvez toujours appeler l'événement **Click** du formulaire en appelant la méthode OnClick.

Par défaut, un événement défini en Visual Basic déclenche ses gestionnaires d'événements suivant l'ordre dans lequel les connexions sont établies. Dans la mesure où les événements peuvent comporter des paramètres **ByRef**, un processus qui se connecte ultérieurement peut recevoir des paramètres ayant été modifiés par un gestionnaire d'événements précédent. Une fois que les gestionnaires d'événements s'exécutent, le contrôle est retourné à la sous-routine qui a déclenché l'événement.

Remarque

Les événements non partagés ne doivent pas être déclenchés dans le constructeur de la classe dans laquelle ils sont déclarés. Bien que de tels événements ne causent pas d'erreurs à l'exécution, ils peuvent ne pas être détectés par des gestionnaires d'événements associés. Utilisez le modificateur **Shared** pour créer un événement partagé si vous avez besoin de déclencher un événement à partir d'un constructeur.

Vous pouvez modifier le comportement par défaut des événements en définissant un événement personnalisé. Pour les événements personnalisés, l'instruction **RaiseEvent** appelle l'accesseur **RaiseEvent** de l'événement. Pour plus d'informations sur les événements personnalisés, consultez Event, instruction.

Exemple

L'exemple suivant utilise des événements pour commencer le compte à rebours des secondes de 10 à 0. Le code illustre de nombreuses méthodes, propriétés et instructions relatives aux événements, y compris l'instruction **RaiseEvent**.

La classe qui déclenche un événement est la source de l'événement, et les méthodes qui traitent l'événement sont les

gestionnaires d'événements. Une source d'événement peut avoir des gestionnaires multiples pour les événements qu'elle génère. Lorsque la classe génère l'événement, celui-ci se produit pour chaque classe ayant choisi de gérer les événements pour cette instance de l'objet.

Cet exemple utilise aussi un formulaire (`Form1`) comportant un bouton (`Button1`) et une zone de texte (`TextBox1`). Lorsque vous cliquez sur le bouton, la première zone affiche un compte à rebours de 10 à 0 secondes. Lorsque la durée totale (10 secondes) s'est écoulée, la première zone de texte affiche « Terminé ».

Le code correspondant à `Form1` indique les états de début et de fin du formulaire. Il contient également le code exécuté lors du déclenchement des événements.

Pour utiliser cet exemple, ouvrez un nouveau projet d'application Windows, ajoutez un bouton appelé `Button1` et une zone de texte appelée `TextBox1` au formulaire principal appelé `Form1`. Cliquez ensuite avec le bouton droit sur le formulaire, puis cliquez sur **Afficher le code** pour ouvrir l'éditeur de code.

Ajoutez une variable **WithEvents** à la section des déclarations de la classe `Form1`.

VB

```
Private WithEvents mText As TimerState
```

Ajoutez le code suivant au code pour `Form1`. Remplacez toute procédure en double éventuelle, telle que `Form_Load` ou `Button_Click`.

VB

```
Private Sub Form1_Load(ByVal sender As Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Button1.Text = "Start"
    mText = New TimerState
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles Button1.Click
    mText.StartCountdown(10.0, 0.1)
End Sub

Private Sub mText_ChangeText() Handles mText.Finished
    TextBox1.Text = "Done"
End Sub

Private Sub mText_UpdateTime(ByVal Countdown As Double) _
                             Handles mText.UpdateTime
    TextBox1.Text = Format(Countdown, "##0.0")
    ' Use DoEvents to allow the display to refresh.
    My.Application.DoEvents()
End Sub

Class TimerState
    Public Event UpdateTime(ByVal Countdown As Double)
    Public Event Finished()
    Public Sub StartCountdown(ByVal Duration As Double, _
                              ByVal Increment As Double)
        Dim Start As Double = DateAndTime.Timer
        Dim ElapsedTime As Double = 0

        Dim SoFar As Double = 0
        Do While ElapsedTime < Duration
            If ElapsedTime > SoFar + Increment Then
                SoFar += Increment
                RaiseEvent UpdateTime(Duration - SoFar)
            End If
            ElapsedTime = DateAndTime.Timer - Start
        Loop
        RaiseEvent Finished()
    End Sub
End Class
```

Appuyez sur F5 pour exécuter l'exemple précédent, puis cliquez sur le bouton **Démarrer**. La première zone de texte commence le compte à rebours des secondes. Lorsque la durée totale (10 secondes) s'est écoulée, la première zone de texte affiche « Terminé ».

Remarque

La méthode **My.Application.DoEvents** ne traite pas les événements exactement de la même manière que le formulaire. Pour permettre au formulaire de gérer les événements directement, vous pouvez utiliser le multithreading. Pour plus d'informations, consultez Multithreading dans Visual Basic.

Voir aussi

Tâches

Comment : ajouter des événements à une classe

Référence

Event, instruction

AddHandler, instruction

RemoveHandler, instruction

Handles

Autres ressources

Événements en Visual Basic

ReDim, instruction (Visual Basic)

Réaffecte l'espace de stockage pour une variable tableau.

```
ReDim [ Preserve ] name(boundlist) [ , name(boundlist) [, ... ] ]
```

Éléments

Preserve

Facultatif. Modificateur utilisé pour conserver les données du tableau existant lorsque vous changez la taille de la dernière dimension seulement.

name

Obligatoire. Nom de la variable tableau. Consultez Noms d'éléments déclarés.

boundlist

Obligatoire. Liste des limites de chaque dimension du tableau redéfini.

Notes

Vous pouvez utiliser l'instruction **ReDim** pour changer la taille d'une ou plusieurs dimensions d'un tableau qui a déjà été déclaré. Si vous disposez d'un grand tableau et que certains de ses éléments ne sont plus nécessaires, **ReDim** peut libérer de la mémoire en réduisant la taille du tableau. En revanche, si votre code détermine qu'un tableau doit contenir plus d'éléments, **ReDim** peut les ajouter.

L'instruction **ReDim** est destinée uniquement aux tableaux. Elle n'est pas valide pour les fonctions scalaires (variables contenant uniquement une seule valeur), les collections ou les structures. Notez que si vous déclarez une variable comme étant de type **Array**, l'instruction **ReDim** ne dispose pas d'informations de type suffisantes pour créer le nouveau tableau.

Vous pouvez utiliser **ReDim** seulement au niveau de la procédure. Cela signifie que le *contexte de déclaration* pour une variable doit être une procédure, et ne peut pas être un fichier source, un espace de noms, une interface, une classe, une structure, un module ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Règles

- **Modificateurs.** Vous pouvez spécifier uniquement le modificateur **Preserve**, et vous ne pouvez pas omettre le mot clé **ReDim** dans ce cas.
- **Variables multiples.** Vous pouvez redimensionner plusieurs variables tableau dans la même instruction de déclaration, en spécifiant les éléments *name* et *boundlist* pour chacune d'elles. Les variables multiples sont séparées par des virgules.
- **Limites de tableau.** Chaque entrée dans *boundlist* peut spécifier les limites inférieures et supérieures de cette dimension. La limite inférieure est toujours zéro, que vous la spécifiez ou pas. La limite supérieure est la valeur la plus élevée possible pour cet indice, et non la longueur de la dimension (qui représente la limite supérieure plus un). Chaque indice peut être compris entre zéro et sa valeur de limite supérieure.

  Le nombre de dimensions de *boundlist* doit correspondre au rang d'origine du tableau.
- **Tableaux vides.** Il est possible d'utiliser la valeur -1 pour déclarer la limite supérieure d'une dimension d'un tableau. Cela signifie que le tableau est vide, mais pas Nothing (Visual Basic). Pour plus d'informations, consultez Comment : créer un tableau sans éléments. Toutefois, le code Visual Basic ne peut pas accéder à un tel tableau. Si vous tentez d'y accéder, une erreur IndexOutOfRangeException se produit lors de l'exécution.
- **Types de données.** L'instruction **ReDim** ne peut pas modifier le type de données d'une variable tableau ou de ses éléments.
- **Initialisation.** L'instruction **ReDim** ne peut pas fournir de nouvelles valeurs d'initialisation pour les éléments du tableau.
- **Rang.** L'instruction **ReDim** ne peut pas changer le rang (nombre de dimensions) du tableau.
- **Redimensionnement avec conservation.** Si vous utilisez **Preserve**, vous pouvez redimensionner seulement la dernière dimension du tableau. Pour toutes les autres dimensions, vous devez spécifier la même limite que dans le tableau existant.

Par exemple, si votre tableau ne comporte qu'une seule dimension, vous pouvez la modifier tout en conservant le contenu du tableau, car vous changez la dernière et seule dimension. Toutefois, si votre tableau comporte deux dimensions ou plus, vous pouvez changer la taille de la dernière dimension seulement si vous utilisez **Preserve**.

- **Propriétés.** Vous pouvez utiliser **ReDim** sur une propriété qui contient un tableau de valeurs.

Comportement

- **Remplacement de tableau. ReDim** libère le tableau existant et crée un nouveau tableau avec le même rang. Le nouveau tableau remplace le tableau libéré dans la variable tableau.
- **Initialisation sans conservation.** Si vous ne spécifiez pas **Preserve**, **ReDim** initialise les éléments du nouveau tableau sur la valeur par défaut de leur type de données.
- **Initialisation avec conservation.** Si vous spécifiez le modificateur **Preserve**, Visual Basic copie les éléments à partir du tableau existant dans le nouveau tableau.

Exemple

L'exemple suivant augmente la taille de la dernière dimension d'un tableau dynamique sans perte de données existantes dans le tableau, puis réduit la taille avec une perte partielle de données. Enfin, il réduit la taille à sa valeur d'origine et réinitialise tous les éléments du tableau.

VB

```
Dim intArray(10, 10, 10) As Integer
ReDim Preserve intArray(10, 10, 20)
ReDim Preserve intArray(10, 10, 15)
ReDim intArray(10, 10, 10)
```

La première instruction **ReDim** crée un nouveau tableau qui remplace le tableau existant dans la variable `intArray`. **ReDim** copie tous les éléments du tableau existant dans le nouveau tableau. Elle ajoute également 10 autres colonnes à la fin de chaque ligne dans chaque couche et initialise les éléments contenus dans ces nouvelles colonnes sur 0 (la valeur par défaut de **Integer**, le type d'élément du tableau).

La deuxième instruction **ReDim** crée un autre nouveau tableau en copiant tous les éléments qui coïncident. Toutefois, cinq colonnes sont perdues à partir de la fin de chaque ligne de chaque couche. Cela n'est pas un problème si vous avez terminé d'utiliser ces colonnes. Réduire la taille d'un grand tableau peut libérer de la mémoire dont vous n'avez plus besoin.

La troisième instruction **ReDim** crée toujours un autre nouveau tableau, en supprimant cinq autres colonnes de la fin de chaque ligne dans chaque couche. Cette fois-ci, elle ne copie pas les éléments existants. Cela permet de rétablir le tableau à sa taille d'origine et de retourner tous ses éléments à leur valeur d'origine par défaut.

Voir aussi

Tâches

Comment : créer un tableau sans éléments

Référence

Const, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Erase, instruction (Visual Basic)
Nothing (Visual Basic)
IndexOutOfRangeException

REM, instruction (Visual Basic)

Utilisée pour inclure des notes explicatives dans le code source d'un programme.

```
REM comment
' comment
```

Éléments
comment

Facultatif. Texte du commentaire de votre choix que vous voulez inclure. Un espace est requis entre le mot clé **REM** ou guillemet simple (') et *comment*.

Notes

Vous pouvez mettre une instruction **REM** toute seule sur une ligne ou bien vous pouvez la mettre sur une ligne qui suit une autre instruction. L'instruction **REM** doit être la dernière instruction de la ligne. Si elle suit une autre instruction, **REM** doit être séparée de cette instruction par un espace.

Vous pouvez utiliser un guillemet simple (') à la place de **REM**. Cela est vrai si votre commentaire suit une autre instruction sur la même ligne ou s'il se trouve tout seul sur une ligne.

Remarque

Vous ne pouvez pas continuer une instruction **REM** à l'aide d'une séquence de continuation de ligne (_). Une fois le commentaire commencé, le compilateur n'examine plus les caractères pour y rechercher une signification particulière. Lorsqu'un commentaire tient sur plusieurs lignes, utilisez une autre instruction **REM** ou un symbole de commentaire (') sur chaque ligne.

Exemple

L'exemple suivant illustre l'instruction **REM** qui est utilisée pour inclure des notes explicatives dans un programme. Il montre également qu'il est possible d'utiliser le guillemet simple (') à la place de **REM**.

VB

```
Dim demoStr1, demoStr2 As String
demoStr1 = "Hello" REM Comment after a statement using REM.
demoStr2 = "Goodbye" ' Comment after a statement using the ' character.
REM This entire line is a comment.
' This entire line is also a comment.
```

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Concepts

Commentaires dans le code

RemoveHandler, instruction

Supprime l'association qui lie un événement et un gestionnaire d'événements.

```
RemoveHandler event, AddressOf eventhandler
```

Éléments

event

Nom de l'événement géré.

eventhandler

Nom de la procédure qui gère actuellement l'événement.

Notes

Les instructions **AddHandler** et **RemoveHandler** vous permettent de débuter et d'arrêter la gestion d'un événement particulier à tout moment au cours de l'exécution du programme.

Pour les événements personnalisés, l'instruction **RemoveHandler** appelle l'accesseur **RemoveHandler** de l'événement. Pour plus d'informations sur les événements personnalisés, consultez Event, instruction.

Exemple

VB

```
Sub TestEvents()
    Dim Obj As New Class1
    ' Associate an event handler with an event.
    AddHandler Obj.Ev_Event, AddressOf EventHandler
    ' Call the method to raise the event.
    Obj.CauseSomeEvent()
    ' Stop handling events.
    RemoveHandler Obj.Ev_Event, AddressOf EventHandler
    ' This event will not be handled.
    Obj.CauseSomeEvent()
End Sub

Sub EventHandler()
    ' Handle the event.
    MsgBox("EventHandler caught event.")
End Sub

Public Class Class1
    ' Declare an event.
    Public Event Ev_Event()
    Sub CauseSomeEvent()
        ' Raise an event.
        RaiseEvent Ev_Event()
    End Sub
End Class
```

Voir aussi

Référence

AddHandler, instruction

Handles

Event, instruction

Concepts

Événements et gestionnaires d'événements

AddHandler et RemoveHandler

Resume, instruction

Reprend l'exécution dès la fin d'une routine de gestion des erreurs.

```
Resume [ Next | line ]
```

Éléments

Resume

Obligatoire. Si l'erreur s'est produite dans la même procédure que le gestionnaire d'erreurs, l'exécution reprend par l'instruction qui a engendré l'erreur. Si l'erreur s'est produite dans une procédure appelée, l'exécution reprend par la dernière instruction qui a appelé la procédure contenant la routine de gestion des erreurs.

Next

Facultatif. Si l'erreur s'est produite dans la même procédure que le gestionnaire d'erreurs, l'exécution reprend par l'instruction qui suit celle qui a engendré l'erreur. Si l'erreur s'est produite dans une procédure appelée, l'exécution reprend par l'instruction qui suit celle qui a appelé la procédure contenant la routine de gestion des erreurs (ou l'instruction **On Error Resume Next**).

line

Facultatif. L'exécution reprend à la ligne spécifiée dans l'argument *line* requis. L'argument *line* est une étiquette de ligne ou un numéro de ligne et doit se trouver dans la même procédure que le gestionnaire d'erreurs.

Notes

Si vous utilisez une instruction **Resume** à un autre endroit que dans la routine de gestion des erreurs, une erreur se produit.

L'instruction **Resume** ne peut pas être utilisée dans toutes les procédures qui contiennent une instruction **Try...Catch...Finally**.

Exemple

Cet exemple utilise l'instruction **Resume** pour terminer la gestion des erreurs dans une procédure, puis reprend l'exécution par l'instruction qui a généré l'erreur. L'erreur numéro 55 est générée pour illustrer l'utilisation de l'instruction **Resume**.

VB

```
Sub ResumeStatementDemo()
  On Error GoTo ErrorHandler   ' Enable error-handling routine.
  Dim x As Integer = 32
  Dim y As Integer = 0
  Dim z As Integer
  z = x / y   ' Creates a divide by zero error
  Exit Sub   ' Exit Sub to avoid error handler.
ErrorHandler:     ' Error-handling routine.
  Select Case Err.Number   ' Evaluate error number.
      Case 6   ' "Divide by zero" error.
        y = 1 ' Sets the value of y to 1 and tries the calculation again.
      Case Else
        ' Handle other situations here....
  End Select
  Resume   ' Resume execution at same line
  ' that caused the error.
End Sub
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Assembly : bibliothèque Runtime Visual Basic (dans Microsoft.VisualBasic.dll)

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Error, instruction
On Error, instruction (Visual Basic)

Return, instruction (Visual Basic)

Retourne le contrôle au code qui a appelé une procédure **Function**, **Sub**, **Get**, **Set** ou **Operator**.

```
Return
-or-
Return expression
```

Élément
expression

Obligatoire dans une procédure **Function**, **Get** ou **Operator**. Expression qui représente la valeur à retourner au code appelant.

Notes

Dans une procédure **Sub** ou **Set**, l'instruction **Return** est équivalente à une instruction **Exit Sub Exit Property**, et *expression* ne doit pas être fourni.

Dans une procédure **Function**, **Get** ou **Operator**, l'instruction **Return** doit contenir *expression*, et *expression* doit avoir pour valeur un type de données convertible vers le type de retour de la procédure. Dans une procédure **Function** ou **Get**, vous pouvez également assigner une expression au nom de la procédure pour servir de valeur de retour, puis exécuter une instruction **Exit Function** ou **Exit Property**. Dans une procédure **Operator**, vous devez utiliser **Return** *expression*.

Vous pouvez inclure autant d'instructions **Return** que nécessaire dans la même procédure.

Remarque

Le code d'un bloc **Finally** est exécuté après que le programme rencontre une instruction **Return** dans un bloc **Try** ou **Catch**, mais avant que cette instruction **Return** ne soit exécutée. Dans ce cas, une instruction **Return** figurant dans le bloc **Finally** es t exécutée *avant* l'instruction **Return** initiale, ce qui permet d'avoir une valeur de retour différente. Pour éviter cette situation potentiellement source de confusion, évitez d'utiliser des instructions **Return** dans les blocs **Finally**.

Exemple

L'exemple suivant utilise plusieurs fois l'instruction **Return** pour retourner au code appelant lorsque la procédure ne doit pas effectuer d'autres opérations.

VB

```
Public Function getAgePhrase(ByVal age As Integer) As String
    If age > 60 Then Return "Senior"
    If age > 40 Then Return "Middle-aged"
    If age > 20 Then Return "Adult"
    If age > 12 Then Return "Teen-aged"
    If age > 4 Then Return "School-aged"
    If age > 1 Then Return "Toddler"
    Return "Infant"
End Function
```

Voir aussi

Référence

Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Get, instruction
Set, instruction (Visual Basic)
Operator, instruction
Property, instruction
Exit, instruction (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)

Select...Case, instruction (Visual Basic)

Exécute un groupe d'instructions parmi plusieurs autres, selon la valeur d'une expression.

```
Select [ Case ] testexpression
    [ Case expressionlist
        [ statements ] ]
    [ Case Else
        [ elsestatements ] ]
End Select
```

Éléments

testexpression

Obligatoire. Expression. Doit avoir pour valeur l'un des types de données élémentaires (**Boolean**, **Byte**, **Char**, **Date**, **Double**, **Decimal**, **Integer**, **Long**, **Object**, **SByte**, **Short**, **Single**, **String**, **UInteger**, **ULong** et **UShort**).

expressionlist

Obligatoire dans une instruction **Case**. Liste des clauses d'expression représentant les valeurs correspondantes pour *testexpression*. Plusieurs clauses d'expression sont séparées par des virgules. Chaque clause peut prendre l'une des formes suivantes :

- *expression1* **To** *expression2*
- [**Is**] *comparisonoperator expression*
- *expression*

Le mot clé **To** permet de définir les limites d'une plage de valeurs correspondantes pour *testexpression*. La valeur de *expression1* doit être inférieure ou égale à la valeur de *expression2*.

Utilisez le mot clé **Is** avec un opérateur de comparaison (**=**, **<>**, **<**, **<=**, **>** ou **>=**) pour spécifier une restriction sur les valeurs correspondantes pour *testexpression*. Si le mot clé **Is** n'est pas indiqué, il est automatiquement inséré avant *comparisonoperator*.

La syntaxe indiquant uniquement *expression* est traitée comme un cas spécial de la syntaxe **Is** où *comparisonoperator* représente le signe égal (**=**). Cette syntaxe est évaluée en tant que *testexpression* = *expression*.

Les expressions de *expressionlist* peuvent correspondre à n'importe quel type de données, si elles peuvent être implicitement converties au type de *testexpression* et si l'argument *comparisonoperator* correspondant est valide pour les deux types avec lesquels il est utilisé.

statements

Facultatif. Une ou plusieurs instructions qui suivent **Case** s'exécutent si *testexpression* correspond à une clause dans *expressionlist*.

elsestatements

Facultatif. Une ou plusieurs instructions qui suivent **Case Else** s'exécutent si *testexpression* ne correspond pas à une clause dans *expressionlist* de l'une des instructions **Case**.

End Select

Met fin à la définition de la construction **Select**...**Case**.

Notes

Si *testexpression* correspond à une clause **Case** *expressionlist*, les instructions qui suivent cette instruction **Case** s'exécutent dans la prochaine instruction **Case**, **Case Else** ou **End Select**. L'instruction qui suit **End Select** prend ensuite le contrôle. Si *testexpression* correspond à une clause *expressionlist* dans plusieurs clauses **Case**, seules les instructions qui suivent la première correspondance s'exécutent.

L'instruction **Case Else** permet d'indiquer que *elsestatements* doit être exécuté si aucune correspondance n'existe entre *testexpression* et une clause *expressionlist* dans l'une des autres instructions **Case**. Bien que cela ne soit pas indispensable, la

présence d'une instruction **Case Else** dans votre construction **Select Case** peut être utile lorsque *testexpression* prend des valeurs inattendues. Si aucune clause **Case** *expressionlist* ne correspond à *testexpression* et si aucune instruction **Case Else** n'existe, le contrôle passe à l'instruction après **End Select**.

Vous pouvez utiliser plusieurs expressions ou plages dans chaque clause **Case**. Par exemple, la ligne suivante est valide :

```
Case 1 To 4, 7 To 9, 11, 13, Is > maxNumber
```

Remarque

Le mot clé **Is** utilisé dans les instructions **Case** et **Case Else** est identique à Is, opérateur (Visual Basic), qui est utilisé pour la comparaison de référence d'objet.

Vous pouvez indiquer des plages et des expressions multiples pour des chaînes de caractères. Dans l'exemple suivant, **Case** correspond à une chaîne qui est exactement égale à "apples", qui a une valeur entre "nuts" et "soup" dans l'ordre alphabétique ou qui contient la même valeur exacte que la valeur actuelle de `testItem`.

```
Case "apples", "nuts" To "soup", testItem
```

La définition de **Option Compare** peut affecter les comparaisons de chaînes. Sous **Option Compare Text**, les chaînes "Apples" et "apples" sont considérées comme égales, mais pas sous **Option Compare Binary**.

Remarque

Une instruction **Case** contenant plusieurs clauses peut exposer un comportement appelé *court-circuit*. Visual Basic évalue les clauses de la gauche vers la droite, et si une correspondance existe avec *testexpression*, les autres clauses ne sont pas évaluées. Le court-circuit peut améliorer les performances, mais il peut engendrer des résultats inattendus si chaque expression dans *expressionlist* doit être évaluée. Pour plus d'informations sur le court-circuit, consultez Expressions booléennes.

Si le code contenu dans un bloc d'instruction **Case** ou **Case Else** ne doit pas exécuter d'autres instructions du bloc, il peut quitter le bloc en utilisant l'instruction **Exit Select** L'instruction qui suit **End Select** prend alors immédiatement le contrôle.

Les constructions **Select Case** peuvent être imbriquées. Chaque construction **Select Case** imbriquée doit avoir une instruction **End Select** correspondante et doit être entièrement contenue dans un bloc d'instruction **Case** ou **Case Else** unique de la construction **Select Case** externe dans laquelle elle est imbriquée.

Exemple

L'exemple ci-dessous utilise une construction **Select Case** pour écrire une ligne correspondant à la valeur de la variable `number`. La seconde instruction **Case** contient la valeur qui correspond à la valeur actuelle de `number`. Ainsi, l'instruction qui écrit « Entre 6 et 8 inclus » est exécutée.

VB

```
Dim number As Integer = 8
Select Case number
    Case 1 To 5
        Debug.WriteLine("Between 1 and 5, inclusive")
        ' The following is the only Case clause that evaluates to True.
    Case 6, 7, 8
        Debug.WriteLine("Between 6 and 8, inclusive")
    Case 9 To 10
        Debug.WriteLine("Equal to 9 or 10")
    Case Else
        Debug.WriteLine("Not between 1 and 10, inclusive")
End Select
```

Voir aussi

Référence

Choose, fonction
End, instruction
If...Then...Else, instruction (Visual Basic)
Option Compare, instruction
Exit, instruction (Visual Basic)

Set, instruction (Visual Basic)

Déclare une procédure de propriété **Set** utilisée pour assigner une valeur à une propriété.

```
[ <attributelist> ] [ accessmodifier ] Set (ByVal value [ As datatype ])
    [ statements ]
End Set
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif pour, au plus, l'une des instructions **Get** et **Set** de cette propriété. Il peut s'agir de l'une des valeurs suivantes :

- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

value

Obligatoire. Paramètre contenant la nouvelle valeur de la propriété.

datatype

Requis si **Option Strict** a la valeur **On**. Type de données du paramètre *value*. Le type de données spécifié doit être identique à celui de la propriété si cette instruction **Set** est déclarée.

statements

Facultatif. Une ou plusieurs instructions qui s'exécutent lors de l'appel de la procédure de propriété **Set**.

End Set

Obligatoire. Met fin à la définition de la procédure de propriété **Set**.

Notes

Chaque propriété doit contenir une procédure de propriété **Set** à moins que la propriété soit marquée **ReadOnly**. La procédure **Set** est utilisée pour définir la valeur de la propriété.

Visual Basic appelle automatiquement la procédure **Set** d'une propriété lorsqu'une instruction d'assignation fournit une valeur à stocker dans la propriété.

Visual Basic passe un paramètre à la procédure **Set** pendant les assignations de propriétés. Si vous ne précisez pas de paramètre pour **Set**, l'environnement de développement intégré (IDE) utilise un paramètre implicite appelé `value`. Ce paramètre contient la valeur à assigner à la propriété. Vous enregistrez généralement cette valeur dans une variable locale privée et la retournez à chaque appel de la procédure **Get**.

Le corps de la déclaration de propriété peut contenir uniquement les procédures **Get** et **Set** de la propriété entre l'instruction Property et **End Property**. Il ne peut stocker rien d'autre que ces procédures. En particulier, il ne peut pas stocker la valeur actuelle de la propriété. Vous devez stocker cette valeur à l'extérieur de la propriété, car si vous la stockez à l'intérieur de l'une ou l'autre des procédures de propriété, l'autre procédure de propriété ne peut pas y accéder. L'approche habituelle consiste à stocker la valeur dans une variable Private (Visual Basic) déclarée au même niveau que la propriété. Vous devez définir une procédure **Set** dans la propriété à laquelle elle s'applique.

La procédure **Set** prend par défaut le niveau d'accès de sa propriété conteneur à moins que vous utilisiez *accessmodifier* dans l'instruction **Set**.

Règles

- **Niveaux d'accès mixtes.** Si vous définissez une propriété en lecture-écriture, vous pouvez éventuellement spécifier un niveau d'accès différent pour la procédure **Get** ou **Set**, mais pas pour les deux. Dans ce cas, le niveau d'accès de la procédure doit être plus restrictif que celui de la propriété. Par exemple, si la propriété est déclarée **Friend**, vous pouvez déclarer la procédure **Set Private**, mais pas **Public**.

  Si vous définissez une propriété **WriteOnly**, la procédure **Set** représente l'ensemble de la propriété. Vous ne pouvez pas déclarer un niveau d'accès différent pour **Set**, parce que deux niveaux d'accès seront définis pour la propriété.

Comportement

- **Retour d'une procédure de propriété.** Lorsque la procédure **Set** retourne au code appelant, l'exécution continue à suivre l'instruction qui a fourni la valeur à stocker.

  Les procédures de propriété **Set** peuvent être retournées à l'aide de Return, instruction (Visual Basic) ou de Exit, instruction (Visual Basic).

  Les instructions **Exit Property** et **Return** provoquent la sortie immédiate d'une procédure de propriété. Plusieurs instructions **Exit Property** et **Return** peuvent apparaître n'importe où dans la procédure, et vous pouvez associer des instructions **Exit Property** et **Return**.

Exemple

L'exemple suivant utilise l'instruction **Set** pour définir la valeur d'une propriété.

VB

```
Class propClass
    Private propVal As Integer
    Property prop1() As Integer
        Get
            Return propVal
        End Get
        Set(ByVal value As Integer)
            propVal = value
        End Set
    End Property
End Class
```

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Référence

Get, instruction

Property, instruction

Sub, instruction (Visual Basic)

Stop, instruction (Visual Basic)

Interrompt l'exécution.

```
Stop
```

Notes

Vous pouvez placer des instructions **Stop** n'importe où dans les procédures afin d'en interrompre l'exécution. Utiliser l'instruction **Stop** revient à placer un point d'arrêt dans le code.

L'instruction **Stop** interrompt l'exécution, mais contrairement à **End**, elle ne ferme pas les fichiers et n'efface pas les variables, sauf si elle est placée dans un fichier exécutable (.exe) compilé.

Remarque

Si l'instruction **Stop** figure dans le code qui s'exécute en dehors de l'environnement de développement intégré (IDE, Integrated Development Environment), le débogueur est appelé. C'est le cas quel que soit le mode de compilation du code (mode débogage ou mode commercial).

Exemple

Cet exemple utilise l'instruction **Stop** pour suspendre l'exécution à chaque itération de la boucle **For...Next**.

VB

```
Dim i As Integer
For i = 1 To 10
    Debug.WriteLine(i)
    ' Stop during each iteration and wait for user to resume.
    Stop
Next i
```

Voir aussi

Référence

End, instruction

Structure, instruction

Déclare le nom d'une structure et introduit la définition des variables, propriétés, événements et procédures contenus dans la structure.

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ Partial ] _
Structure name [ ( Of typelist ) ]
    [ Implements interfacenames ]
        datamemberdeclarations
    [ methodmemberdeclarations ]
End Structure
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

Shadows

Facultatif. Consultez Shadows.

Partial

Facultatif. Indique une définition partielle de la structure. Consultez Partial (Visual Basic).

name

Obligatoire. Nom de cette structure. Consultez Noms d'éléments déclarés.

Of

Facultatif. Indique qu'il s'agit d'une structure générique.

typelist

Obligatoire si vous utilisez le mot clé Of. Liste de paramètres de type pour cette structure. Consultez Liste de types.

Implements

Facultatif. Indique que cette structure implémente les membres d'une ou plusieurs interfaces. Consultez l'instruction Implements.

interfacenames

Obligatoire si vous utilisez l'instruction **Implements**. Noms des interfaces que cette structure implémente.

datamemberdeclarations

Obligatoire. Une ou plusieurs instructions **Const**, **Dim**, **Enum** ou **Event** qui déclarent des *données membres* de la structure.

methodmemberdeclarations

Facultatif. Zéro ou plusieurs déclarations des procédures **Function**, **Operator**, **Property** ou **Sub** utilisées comme *membres de méthode* de la structure.

End Structure

Obligatoire. Met fin à la définition **Structure**.

Notes

L'instruction **Structure** définit un type valeur composite que vous pouvez personnaliser. Une *structure* est une généralisation du type défini par l'utilisateur (UDT, User-Defined Type) des précédentes versions de Visual Basic. Pour plus d'informations, consultez Structures constituées à partir de vos propres types de données.

Les structures prennent en charge un grand nombre de mêmes fonctionnalités comme classes. Par exemple, les structures peuvent avoir des propriétés et des procédures, implémenter des interfaces et posséder des constructeurs paramétrés. Toutefois, des différences importantes existent entre les structures et les classes dans des domaines tels que l'héritage, les déclarations et l'utilisation. De même, les classes sont des types référence et les structures des types valeur. Pour plus d'informations, consultez Structures et classes.

Vous pouvez utiliser **Structure** uniquement au niveau de l'espace de noms ou du module. Cela signifie que le *contexte de déclaration* pour une structure doit être un fichier source, un espace de noms, une classe, une structure, un module ou une interface, et ne peut pas être une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les structures disposent par défaut d'un accès Friend (Visual Basic). Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès. Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Règles

- **Imbrication.** Vous pouvez définir une structure dans une autre. La structure externe est appelée *structure conteneur*, et la structure interne est appelée *structure imbriquée*. Toutefois, vous ne pouvez pas accéder aux membres d'une structure imbriquée par l'intermédiaire de la structure conteneur. Vous devez plutôt déclarer une variable du type de données de la structure imbriquée.
- **Déclaration de membre.** Vous devez déclarer chaque membre d'une structure. Un membre d'une structure ne peut pas être Protected ou **Protected Friend**, car rien ne peut être hérité d'une structure Toutefois, la structure proprement dite peut être **Protected** ou **Protected Friend**.

  Vous devez déclarer au moins une variable non partagée ou un événement non personnalisé et non partagé dans une structure. Vous ne pouvez pas posséder uniquement des constantes, propriétés et procédures, même si certaines sont non partagées.
- **Initialisation.** Vous ne pouvez pas initialiser la valeur d'une donnée membre non partagée d'une structure comme élément de sa déclaration. Vous devez soit initialiser une donnée membre au moyen d'un constructeur paramétré sur la structure, soit assigner une valeur au membre après avoir créé une instance de la structure.
- **Héritage.** Une structure ne peut pas hériter d'un type autre que ValueType, d'où héritent toutes les structures. En particulier, une structure ne peut pas hériter d'une autre.

  Vous ne pouvez pas utiliser l'instruction Inherits dans une définition de structure, même pour spécifier **ValueType**.
- **Implémentation.** Si la structure utilise l'instruction Implements, vous devez implémenter chaque membre défini par chaque interface que vous spécifiez dans *interfacenames*.
- **Propriété par défaut.** Une structure peut spécifier au plus une propriété comme sa *propriété par défaut*, à l'aide du modificateur Default (Visual Basic). Pour plus d'informations, consultez Propriétés par défaut.

Comportement

- **Niveau d'accès.** Dans une structure, vous pouvez déclarer chaque membre avec son propre niveau d'accès. Tous les membres de structure disposent par défaut d'un accès Public (Visual Basic). Notez que si la structure proprement dite a un niveau d'accès plus restreint, cela restreint automatiquement l'accès à ses membres, même si vous modifiez leurs niveaux d'accès à l'aide des modificateurs d'accès.
- **Portée.** Une structure est présente dans la portée dans l'ensemble de son espace de noms, sa classe, sa structure ou son module conteneur.

  La portée d'un membre de structure est la structure entière.
- **Durée de vie.** Une structure elle-même n'a pas de durée de vie. En revanche, chaque instance de cette structure a une durée de vie indépendante de toutes les autres instances.

La durée de vie d'une instance commence à sa création par une clause New (Visual Basic) Elle se termine lorsque la durée de vie de la variable qui la contient prend fin.

Vous ne pouvez pas étendre la durée de vie d'une instance de structure. Une approximation des fonctionnalités de structure statiques est fournie par un module. Pour plus d'informations, consultez Module, instruction.

Les membres de structure ont des durées de vie qui varient selon leur méthode et leur lieu de déclaration. Pour plus d'informations, consultez « Durée de vie » dans Class, instruction (Visual Basic).

- **Qualification.** Le code à l'extérieur d'une structure doit qualifier le nom d'un membre avec le nom de cette structure.

  Si le code à l'intérieur d'une structure imbriquée crée une référence non qualifiée à un élément de programmation, Visual Basic recherche d'abord l'élément dans la structure imbriquée, puis dans sa structure conteneur, et ainsi de suite jusqu'à l'élément conteneur le plus externe. Pour plus d'informations, consultez Résolution d'une référence lorsque plusieurs variables ont le même nom.

- **Consommation de mémoire.** Comme dans le cas de tous les types de données composites, vous ne pouvez pas calculer en toute sécurité la consommation totale de la mémoire d'une structure en additionnant les allocations de stockage nominal de ses membres. De plus, il est risqué de supposer que l'ordre de stockage dans la mémoire est identique à l'ordre de déclaration. Si vous devez contrôler la disposition de stockage d'une structure, vous pouvez appliquer l'attribut StructLayoutAttribute à l'instruction **Structure**.

Exemple

L'exemple suivant utilise l'instruction **Structure** pour définir un ensemble de données apparentées pour un employé. Il montre l'utilisation des membres **Public**, **Friend** et **Private** pour refléter la sensibilité des éléments de données. Il montre également les membres d'une procédure, d'une propriété et d'un événement.

VB

```
Public Structure employee
    ' Public members, accessible from throughout declaration region.
    Public firstName As String
    Public middleName As String
    Public lastName As String
    ' Friend members, accessible from anywhere within the same assembly.
    Friend employeeNumber As Integer
    Friend workPhone As Long
    ' Private members, accessible only from within the structure itself.
    Private homePhone As Long
    Private level As Integer
    Private salary As Double
    Private bonus As Double
    ' Procedure member, which can access structure's private members.
    Friend Sub calculateBonus(ByVal rate As Single)
        bonus = salary * CDbl(rate)
    End Sub
    ' Property member to return employee's eligibility.
    Friend ReadOnly Property eligible() As Boolean
        Get
            Return level >= 25
        End Get
    End Property
    ' Event member, raised when business phone number has changed.
    Public Event changedWorkPhone(ByVal newPhone As Long)
End Structure
```

Voir aussi

Référence

Class, instruction (Visual Basic)
Interface, instruction (Visual Basic)
Module, instruction
Dim, instruction (Visual Basic)
Const, instruction (Visual Basic)
Enum, instruction (Visual Basic)
Event, instruction

Operator, instruction
Property, instruction

Concepts

Structures et classes

Sub, instruction (Visual Basic)

Déclare le nom, les paramètres et le code qui définissent une procédure **Sub**.

```
[ <attributelist> ] [ accessmodifier ] [ proceduremodifiers ] [ Shared ] [ Shadows ]
Sub name [ (Of typeparamlist) ] [ (parameterlist) ] [ Implements implementslist | Handles e
ventlist ]
    [ statements ]
    [ Exit Sub ]
    [ statements ]
End Sub
```

Éléments

attributelist

Facultatif. Consultez Liste d'attributs.

accessmodifier

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Public
- Protégé
- Friend
- Private
- **Protected Friend**

Consultez Niveaux d'accès dans Visual Basic.

proceduremodifiers

Facultatif. Il peut s'agir de l'une des valeurs suivantes :

- Overloads
- Substitutions
- Overridable
- NotOverridable
- MustOverride
- **MustOverride Overrides**
- **NotOverridable Overrides**

Shared

Facultatif. Consultez Partagé.

Shadows

Facultatif. Consultez Shadows.

name

Obligatoire. Nom de la procédure. Consultez Noms d'éléments déclarés.

typeparamlist

Facultatif. Liste de paramètres de type pour une procédure générique. Consultez Liste de types.

parameterlist

Facultatif. Liste des noms de variables locales représentant les paramètres de cette procédure. Consultez Liste de paramètres.

Implements

Facultatif. Indique que cette procédure implémente une ou plusieurs procédures **Sub**, chacune étant définie dans une interface implémentée par la classe ou la structure conteneur de cette procédure. Consultez Implements, instruction.

implementslist

Requis si **Implements** est fourni. Liste de procédures **Sub** implémentées.

```
implementedprocedure [ , implementedprocedure ... ]
```

Chaque *implementedprocedure* emploie la syntaxe et les éléments suivants :

```
interface.definedname
```

Élément	Description
interface	Obligatoire. Nom d'une interface implémentée par la classe ou la structure conteneur de cette procédure.
definedname	Obligatoire. Nom par lequel la procédure est définie dans *interface*.

Handles

Facultatif. Indique que cette procédure peut gérer un ou plusieurs événements spécifiques. Consultez Handles.

eventlist

Requis si **Handles** est fourni. Liste des événements gérés par cette procédure.

```
eventspecifier [ , eventspecifier ... ]
```

Chaque *eventspecifier* emploie la syntaxe et les éléments suivants :

```
eventvariable.event
```

Élément	Description
eventvariable	Obligatoire. Variable objet déclarée avec le type de données de la classe ou de la structure qui déclenche l'événement.
event	Obligatoire. Nom de l'événement géré par cette procédure.

statements

Facultatif. Bloc d'instructions à exécuter dans cette procédure.

End Sub

Met fin à la définition de cette procédure.

Notes

Tout le code exécutable doit se trouver dans une procédure. Utilisez une procédure **Sub** lorsque qu'il n'est pas nécessaire de retourner une valeur au code appelant. Utilisez une procédure **Function** lorsque vous devez retourner une valeur.

Vous pouvez utiliser **Sub** seulement au niveau du module. Cela signifie que le *contexte de déclaration* pour une procédure sub doit être une classe, une structure, un module ou une interface et ne peut pas être un fichier source, un espace de noms, une procédure ou un bloc. Pour plus d'informations, consultez Contextes de déclaration et niveaux d'accès par défaut.

Les procédures **Sub** prennent l'accès public par défaut. Vous pouvez régler leurs niveaux d'accès avec les modificateurs d'accès.

Règles

- **Implémentation.** Si cette procédure utilise le mot clé **Implements**, la classe ou la structure conteneur doit avoir une instruction **Implements** qui suit immédiatement son instruction **Class** ou **Structure**. L'instruction **Implements** doit inclure chaque interface spécifiée dans *implementslist*. Toutefois, le nom par lequel une interface définit le **Sub** (dans *definedname*) ne doit pas être identique à celui de cette procédure (dans *name*).

Comportement

- **Retour d'une procédure.** Lorsque la procédure **Sub** retourne au code appelant, l'exécution se poursuit avec

l'instruction qui suit celle qui l'a appelé.

Les instructions **Exit Sub** et **Return** provoquent la sortie immédiate d'une procédure **Sub**. Un nombre indéterminé d'instructions **Exit Sub** et **Return** peut apparaître n'importe où dans la procédure et vous pouvez mélanger les instructions **Exit Sub** et **Return**.

L'exemple suivant affiche un retour d'une procédure **Sub**.

```
                Sub mySub(ByVal q As String)
    Return
End Sub
```

- **Appel d'une procédure.** Une procédure **Sub**, à l'instar d'une procédure **Function**, est une procédure distincte qui peut prendre en charge des paramètres et exécuter une série d'instructions. Toutefois, contrairement à une procédure **Function**, qui retourne une valeur, une procédure **Sub** ne peut pas être utilisée dans une expression.

  Vous appelez une procédure **Sub** en utilisant le nom de procédure, suivi de la liste d'arguments entre parenthèses, dans une instruction. Vous pouvez omettre les parenthèses seulement si vous n'indiquez aucun argument. Toutefois, votre code est plus lisible si vous incluez toujours les parenthèses.

  Vous pouvez éventuellement utiliser l'instruction **Call** pour appeler une procédure **Sub**. Cela permet d'améliorer la lisibilité de votre code.

Dépannage

Ordre d'exécution. Visual Basic réorganise quelquefois les expressions arithmétiques de manière à optimiser l'efficacité interne. Pour cette raison, si votre liste d'arguments inclut des expressions qui appellent d'autres procédures, ne vous attendez pas à ce qu'elles soient appelées dans un ordre donné.

Exemple

L'exemple suivant utilise l'instruction **Sub** pour définir le nom, les paramètres et le code qui forment le corps d'une procédure **Sub**.

VB

```
Sub computeArea(ByVal length As Double, ByVal width As Double)
    ' Declare local variable.
    Dim area As Double
    If length = 0 Or width = 0 Then
        ' If either argument = 0 then exit Sub immediately.
        Exit Sub
    End If
    ' Calculate area of rectangle.
    area = length * width
    ' Print area to Immediate window.
    Debug.WriteLine(area)
End Sub
```

Voir aussi

Tâches

Comment : utiliser une classe générique

Procédures de dépannage

Référence

Implements, instruction

Function, instruction (Visual Basic)

Liste de paramètres

Dim, instruction (Visual Basic)

Call, instruction (Visual Basic)

Of

Concepts

Tableaux de paramètres

SyncLock, instruction

Acquiert un verrou exclusif pour un bloc d'instruction avant d'exécuter le bloc.

```
SyncLock lockobject
    [ block ]
End SyncLock
```

Éléments
lockobject

Requis. Expression qui prend la valeur d'une référence d'objet.

block

Facultatif. Bloc d'instructions qui doivent être exécutées lorsque le verrou est acquis.

End SyncLock

Met fin à un bloc **SyncLock**.

Notes

L'instruction **SyncLock** garantit que plusieurs threads n'exécutent pas le bloc d'instruction en même temps. **SyncLock** empêche chaque thread d'entrer dans le bloc jusqu'à ce qu'aucun autre thread ne l'exécute.

L'utilisation la plus courante de **SyncLock** est d'empêcher la mise à jour des données par plusieurs threads simultanément. Si les instructions qui manipulent les données doivent être exécutées sans interruption, mettez-les dans un bloc **SyncLock**.

Un bloc d'instruction protégé par un verrou exclusif est parfois appelé une *section critique*.

Règles

- Branchement. Vous ne pouvez pas vous brancher dans un bloc **SyncLock** de l'extérieur du bloc.
- Valeur de l'objet lock. La valeur de *lockobject* ne peut pas être **Nothing**. Vous devez créer l'objet lock avant de l'utiliser dans une instruction **SyncLock**.

  Vous ne pouvez pas modifier la valeur de *lockobject* en exécutant un bloc **SyncLock**. Le mécanisme exige que l'objet lock reste inchangé.

Comportement

- Mécanisme. Lorsqu'un thread atteint l'instruction **SyncLock**, il évalue l'expression *lockobject* et interrompt l'exécution jusqu'à ce qu'il acquière un verrou exclusif sur l'objet retourné par l'expression. Lorsqu'un autre thread atteint l'instruction **SyncLock**, il n'acquiert pas de verrou jusqu'à ce que le premier thread exécute l'instruction **End SyncLock**.
- Données protégées. Si *lockobject* est une variable **Shared**, le verrou exclusif empêche un tread d'exécuter le bloc **SyncLock** dans toute instance de la classe pendant qu'un autre l'exécute. Cela protège des données qui sont partagées par toutes les instances.

  Si *lockobject* est une variable d'instance (non **Shared**), le verrou empêche un thread qui s'exécute dans l'instance en cours d'exécuter le bloc **SyncLock** en même temps qu'un autre thread dans la même instance. Cela protège des données gérées par l'instance.
- Acquisition et libération. Un bloc **SyncLock** se comporte comme une construction **Try...Finally** dans laquelle le bloc **Try** acquiert un verrou exclusif sur *lockobject* et le bloc **Finally** le libère. C'est pourquoi le bloc **SyncLock** garantit la libération du verrou, peu importe comment vous quittez le bloc. Cela est vrai même dans le cas d'une exception non gérée.
- Appels du Framework. Le bloc **SyncLock** acquiert et libère le verrou exclusif en appelant les méthodes **Enter** et **Exit** de la classe **Monitor** dans l'espace de noms System.Threading.

Méthodes de programmation

L'expression *lockobject* doit toujours prendre la valeur d'un objet qui appartient exclusivement à votre classe. Vous devez

déclarer une variable objet **Private** pour protéger des données qui appartiennent à l'instance en cours ou une variable objet **Private Shared** pour protéger des données communes à toutes les instances.

Vous ne devez pas utiliser le mot clé **Me** pour fournir un objet lock aux données de l'instance. Si un code externe à votre classe comporte une référence à une instance de votre classe, il pourrait utiliser cette référence comme un objet lock pour un bloc **SyncLock** complètement différent du vôtre, protégeant des données différentes. De cette manière, votre classe et l'autre classe pourraient s'empêcher l'une l'autre d'exécuter leurs blocs **SyncLock** non liés. Un verrouillage de ce type sur une chaîne peut poser problème puisque tout autre code du processus utilisant la même chaîne partagera le même verrouillage.

De même, vous ne devez pas utiliser la méthode **Me.GetType** pour fournir un objet lock aux données partagées. Cela est dû au fait que **GetType** retourne toujours le même objet **Type** pour un nom de classe donné. Le code externe pourrait appeler **GetType** de votre classe et obtenir le même objet lock que vous utilisez. Les deux classes s'empêcheraient l'une l'autre d'exécuter à leurs blocs **SyncLock**.

Exemple

L'exemple suivant affiche une classe qui gère une liste ordinaire de messages. Il stocke les messages dans un tableau et le dernier élément utilisé de ce tableau dans une variable. La procédure `addAnotherMessage` incrémente le dernier élément et stocke le nouveau message. Ces deux opérations sont protégées par les instructions **SyncLock** et **End SyncLock**, car une fois que le dernier élément a été incrémenté, le nouveau message doit être stocké avant que tout autre thread puisse incrémenter encore le dernier élément.

VB

```
Class simpleMessageList
    Public messagesList() As String = New String(50) {}
    Public messagesLast As Integer = -1
    Private messagesLock As New Object
    Public Sub addAnotherMessage(ByVal newMessage As String)
        SyncLock messagesLock
            messagesLast += 1
            messagesList(messagesLast) = newMessage
        End SyncLock
    End Sub
End Class
```

Si la classe `simpleMessageList` a partagé une liste de messages entre toutes ses instances, alors les `messagesList` et `messagesLast` de variables seraient déclarés comme **Shared**. Dans ce cas, les `messagesLock` de variables doivent être également **Shared**, afin qu'il y ait un objet lock seul utilisé par chaque instance.

Voir aussi

Référence

System.Threading

Monitor

Concepts

Synchronisation des threads

Autres ressources

Multithreading dans Visual Basic

Throw, instruction (Visual Basic)

Lève une exception dans une procédure.

```
Throw [ expression ]
```

Élément
expression

Fournit des informations sur l'exception à lever. Facultatif lorsqu'elle est présente dans une instruction **Catch**, sinon obligatoire.

Notes

L'instruction **Throw** lève une exception que vous pouvez gérer à l'aide du code de gestion structurée des exceptions (**Try**...**Catch**...**Finally**) ou du code de gestion non structurée des exceptions (**On Error GoTo**). Vous pouvez utiliser l'instruction **Throw** pour intercepter les erreurs dans votre code, compte tenu que Visual Basic monte dans la pile des appels jusqu'à ce qu'il trouve le code de gestion des exceptions approprié.

Une instruction **Throw** sans expression peut être utilisée uniquement dans une instruction **Catch**, auquel cas elle lève de nouveau l'exception qui est gérée par l'instruction **Catch**.

L'instruction **Throw** réinitialise la pile d'appel pour l'exception *expression*. Si *expression* n'est pas fournie, la pile d'appel demeure inchangée. Vous pouvez accéder à la pile d'appel pour l'exception à travers la propriété StackTrace.

Exemple

Le code suivant utilise l'instruction **Throw** pour lever une exception :

VB

```
' Throws a new exception.
Throw New System.Exception("An exception has occurred.")
```

Configuration requise

Espace de noms : Microsoft.VisualBasic

Module : Interaction

Assembly : bibliothèque Visual Basic Runtime (dans Microsoft.VisualBasic.dll)

Voir aussi

Tâches

Comment : vérifier l'exception interne d'une exception

Référence

Try...Catch...Finally, instruction (Visual Basic)

On Error, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Vue d'ensemble de la gestion non structurée des exceptions

Try...Catch...Finally, instruction (Visual Basic)

Permet de traiter une partie ou l'ensemble des erreurs possibles pouvant se produire dans un bloc de code donné, tout en continuant à exécuter le code.

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

Éléments

tryStatements

Facultatif. Instructions dans lesquelles une erreur peut se produire. Il peut s'agir d'instructions composées.

Catch

Facultatif. Plusieurs blocs **Catch** sont autorisés. Si une exception se produit pendant le traitement du bloc **Try**, chaque instruction **Catch** est examinée dans l'ordre textuel afin de déterminer si elle gère cette exception, *exception* représentant l'exception qui a été levée.

exception

Facultatif. Tout nom de variable. La valeur initiale de l'argument *exception* est la valeur de l'erreur levée. Cet argument est utilisé avec **Catch** pour spécifier l'erreur interceptée. S'il est omis, l'instruction **Catch** intercepte n'importe quelle exception.

type

Facultatif. Spécifie le type d'un filtre de classe. Si la valeur de l'argument *exception* est l'un des types spécifiés par *type* ou un type dérivé, l'identificateur est lié à l'objet exception.

When

Facultatif. Une instruction **Catch** avec une clause **When** intercepte les exceptions uniquement lorsque *expression* a la valeur **True**. Une clause **When** n'est appliquée qu'après le contrôle du type de l'exception, et *expression* peut faire référence à l'identificateur qui représente l'exception.

expression

Facultatif. Doit pouvoir être converti implicitement en type **Boolean**. Toute expression qui décrit un filtre générique. Généralement utilisé pour effectuer un filtre selon le numéro de l'erreur. Cet argument est utilisé avec le mot clé **When** pour spécifier des circonstances permettant d'intercepter l'erreur.

catchStatements

Facultatif. Instructions permettant de gérer des erreurs se produisant dans le bloc **Try** associé. Il peut s'agir d'une instruction composée.

Exit Try

Facultatif. Mot clé qui décompose la structure **Try...Catch...Finally**. L'exécution reprend par le code qui suit immédiatement l'instruction **End Try**. L'instruction **Finally** sera toujours exécutée. Non autorisé dans les blocs **Finally**.

Finally

Facultatif. Un bloc **Finally** est toujours exécuté quand l'exécution quitte une partie quelconque de l'instruction **Try**.

finallyStatements

Facultatif. Instructions qui sont exécutées à la fin du traitement de toutes les autres erreurs.

End Try

Met fin à la structure **Try...Catch...Finally**.

Notes

Les variables locales d'un bloc **Try** ne sont pas disponibles dans un bloc **Catch**, car ce sont des blocs séparés. Si vous voulez utiliser une variable dans plusieurs blocs, déclarez la variable à l'extérieur de la structure **Try...Catch...Finally**.

Le bloc **Try** contient du code susceptible de contenir des erreurs tandis que le bloc **Catch** contient du code qui peut gérer toute erreur survenue. Si une erreur se produit dans le bloc **Try**, le programme de contrôle est passé à l'instruction **Catch** appropriée pour y être disposé. L'argument *exception* est une instance de la classe **Exception** ou une classe qui dérive de la classe **Exception**. L'instance de la classe **Exception** correspond à l'erreur qui s'est produite dans le bloc **Try**. L'instance contient des informations sur l'erreur, notamment son numéro et son message.

Si une instruction **Catch** ne spécifie pas d'argument *exception*, elle intercepte tout type de système ou d'exception d'application. Vous devez toujours utiliser cette variation comme dernier bloc **Catch** dans la structure **Try...Catch...Finally**, après avoir intercepté toutes les exceptions spécifiques que vous anticipez. Le flux de contrôle ne peut jamais atteindre un bloc **Catch** qui suit un **Catch** sans argument *exception*.

Dans les situations d'un niveau de confiance partiel, comme une application hébergée sur un partage réseau, **Try...Catch...Finally** n'intercepte pas les exceptions de sécurité qui se produisent avant que la méthode contenant l'appel ne soit appelée. L'exemple suivant, s'il est placé sur un partage réseau et exécuté à cet endroit, produit l'erreur : "System.Security.SecurityException: Échec de la demande". Pour plus d'informations sur les exceptions de sécurité, consultez la classe SecurityException.

VB

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
    Try
        Process.Start("http://www.microsoft.com")
    Catch ex As Exception
        MsgBox("Can't load Web page" & vbCrLf & ex.Message)
    End Try
End Sub
```

Dans une telle situation d'un niveau de confiance partiel, il est nécessaire de mettre l'instruction `Process.Start` dans un **Sub** séparé. L'appel effectué initialement au **Sub** échouera, et **Try...Catch** pourra l'intercepter avant que le **Sub** contenant `Process.Start` soit lancé et que l'exception de sécurité soit produite.

Remarque

Si une instruction **Try** ne contient pas au moins un bloc **Catch**, elle doit contenir un bloc **Finally**.

Exemple

L'exemple simplifié ci-dessous illustre l'utilisation de la structure de l'instruction **Try...Catch...Finally**.

VB

```
Public Sub TryExample()
    Dim x As Integer = 5    ' Declare variables.
    Dim y As Integer = 0
    Try                     ' Set up structured error handling.
        x = x \ y           ' Cause a "Divide by Zero" error.
    Catch ex As Exception When y = 0        ' Catch the error.
        Beep()
        MsgBox("You tried to divide by 0.") ' Show an explanatory message.
    Finally
        Beep()              ' This line is executed no matter what.
    End Try
End Sub
```

Voir aussi

Référence

End, instruction

Err, objet (Visual Basic)

Exit, instruction (Visual Basic)
On Error, instruction (Visual Basic)
GoTo, instruction
Exception

Using, instruction (Visual Basic)

Déclare le début d'un bloc **Using** et acquiert éventuellement les ressources système contrôlées par le bloc.

```
Using { resourcelist | resourceexpression }
    [ statements ]
End Using
```

Éléments
resourcelist

Obligatoire si vous ne fournissez pas *resourceexpression*. Liste d'une ou de plusieurs ressources système contrôlées par ce bloc **Using**.

resourceexpression

Obligatoire si vous ne fournissez pas *resourcelist*. Variable ou expression de référence qui fait référence à une ressource système à contrôler par ce bloc **Using**.

statements

Facultatif. Bloc d'instructions exécutées par le bloc **Using**.

End Using

Obligatoire. Met fin à la définition du bloc **Using** et supprime toutes les ressources qu'il contrôle.

Chaque ressource de l'élément *resourcelist* comporte la syntaxe et les éléments suivants :

```
resourcename As New resourcetype [ ( [ arglist ] ) ]
```

- ou -

```
resourcename As resourcetype = resourceexpression
```

Éléments resourcelist
resourcename

Obligatoire. Variable de référence qui fait référence à une ressource système contrôlée par le bloc **Using**.

New

Obligatoire si l'instruction **Using** acquiert la ressource. Si vous avez déjà acquis la ressource, utilisez la deuxième possibilité de syntaxe.

resourcetype

Obligatoire. Classe de la ressource. La classe doit implémenter l'interface IDisposable.

arglist

Facultatif. Liste d'arguments que vous passez au constructeur pour créer une instance de *resourcetype*. Consultez Liste de paramètres.

resourceexpression

Obligatoire. Variable ou expression qui fait référence à une ressource système qui répond aux exigences de *resourcetype*. Si vous utilisez la deuxième possibilité de syntaxe, vous devez acquérir la ressource avant de passer le contrôle à l'instruction **Using**.

Notes

Votre code requiert parfois une ressource non managée, par exemple un handle de fichier, un wrapper COM ou une connexion SQL. Un bloc **Using** garantit la suppression d'une ou de plusieurs de ces ressources lorsque votre code a fini de les utiliser. Cela permet de les mettre à la disposition d'un autre code.

Les ressources managées sont supprimées par le garbage collector (GC) du .NET Framework sans nécessiter de codage supplémentaire de votre part. Vous n'avez pas besoin d'un bloc **Using** pour les ressources managées.

Un bloc **Using** se compose de trois parties : acquisition, utilisation et suppression.

- L'*acquisition* consiste à créer une variable et à l'initialiser pour faire référence à la ressource système. L'instruction **Using** peut acquérir une ou plusieurs ressources, ou vous pouvez acquérir exactement une ressource avant d'entrer dans le bloc et la fournir à l'instruction **Using**. Si vous fournissez *resourceexpression*, vous devez acquérir la ressource avant de passer le contrôle à l'instruction **Using**.
- L'*utilisation* consiste à accéder aux ressources et à les utiliser pour exécuter des opérations. Les instructions entre **Using** et **End Using** représentent l'utilisation des ressources.
- La *suppression* consiste à appeler la méthode **Dispose** de l'objet dans *resourcename*. Elle permet à l'objet d'arrêter correctement sa ressource non managée. L'instruction **End Using** supprime les ressources sous le contrôle du bloc **Using**.

Comportement

Un bloc **Using** se comporte comme une construction **Try**...**Finally** dans laquelle le bloc **Try** utilise les ressources et le bloc **Finally** les supprime. C'est pourquoi le bloc **Using** garantit la suppression des ressources, peu importe comment vous quittez le bloc. Cela est vrai même dans le cas d'une exception non gérée, sauf pour StackOverflowException.

La portée de chaque variable de ressource acquise par l'instruction **Using** est limitée au bloc **Using**.

Si vous spécifiez plusieurs ressources système dans l'instruction **Using**, vous obtenez le même effet que si vous imbriquez des blocs **Using** les uns dans les autres.

Gestion structurée des exceptions dans un bloc Using

Si vous devez gérer une exception qui peut se produire dans le bloc **Using**, vous pouvez lui ajouter une construction **Try**...**Finally** complète. Si vous devez gérer le cas où l'instruction **Using** n'a pas réussi à acquérir une ressource, vous pouvez la tester pour déterminer si *resourcename* a la valeur **Nothing**.

Gestion structurée des exceptions à la place d'un bloc Using

Si vous devez exercer un plus grand contrôle sur l'acquisition des ressources, ou si vous avez besoin de code supplémentaire dans le bloc **Finally**, vous pouvez réécrire le bloc **Using** en tant que construction **Try**...**Finally**. L'exemple suivant présente les structures des constructions **Try** et **Using** qui sont équivalentes dans l'acquisition et la suppression de `resource`.

```
            Using resource As New resourceType
    ' Insert code to work with resource.
End Using
' THE FOLLOWING TRY CONSTRUCTION IS EQUIVALENT TO THE USING BLOCK
Dim resource As New resourceType
Try
    ' Insert code to work with resource.
Catch ex As Exception
    ' Insert code to process exception.
Finally
    ' Insert code to do additional processing before disposing of resource.
    resource.Dispose()
End Try
```

Remarque

Le code contenu dans le bloc **Using** ne doit pas assigner l'objet dans *resourcename* à une autre variable. Lorsque vous quittez le bloc **Using**, la ressource est supprimée, et l'autre variable ne peut pas accéder à la ressource vers laquelle il pointe.

Exemple

L'exemple suivant utilise un bloc **Using** pour acquérir une nouvelle police. Cela garantit que le système appelle la méthode **Dispose** sur la police lorsque votre code quitte le bloc.

VB

```
Public Sub setbigbold(ByVal c As Control)
    Using nf As New System.Drawing.Font("Arial", 12.0F, _
        System.Drawing.FontStyle.Bold)

        c.Font = nf
        c.Text = "This is 12-point Arial bold"
```

```
        End Using
    End Sub
```

Voir aussi

Tâches

Comment : supprimer une ressource système

Référence

Try...Catch...Finally, instruction (Visual Basic)

IDisposable

System.Drawing

Font

While...End While, instruction (Visual Basic)

Exécute une série d'instructions tant qu'une condition donnée a la valeur **True**.

```
While condition
    [ statements ]
    [ Exit While ]
    [ statements ]
End While
```

Éléments

condition

Requis. Expression **Boolean**. Si *condition* est **Nothing**, Visual Basic le traite comme **False**.

statements

Facultatif. Une ou plusieurs instructions suivant **While** et s'exécutant chaque fois que *condition* a la valeur **True**.

Exit While

Facultatif. Transfert le contrôle hors du bloc **While**.

End While

Obligatoire. Termine la définition du bloc **While**.

Notes

Utilisez une structure **While...End While** lorsque vous souhaitez répéter indéfiniment un ensemble d'instructions, tant qu'une condition reste **True**. Si vous souhaitez plus de souplesse à l'égard de l'emplacement où vous testez la condition ou du résultat pour lequel vous la testez, vous préférez certainement Do...Loop, instruction (Visual Basic). Si vous souhaitez répéter plusieurs fois les instructions, For...Next, instruction (Visual Basic) constitue généralement un meilleur choix.

Si *condition* est **True**, toutes les *statements* s'exécutent jusqu'à ce que l'instruction **End While** soit rencontrée. L'instruction **While** prend ensuite le contrôle et l'élément *condition* est à nouveau vérifié. Si *condition* est toujours **True**, le processus est répété. S'il est **False**, le contrôle passe à l'instruction qui suit l'instruction **End While**.

Règles

- **Nature de la condition.** La condition provient généralement d'une comparaison entre deux valeurs, mais elle peut être une expression qui évalue une valeur Boolean, type de données (Visual Basic) (**True** ou **False**). Elle peut comprendre des valeurs d'autres types de données, tels que des types numériques, qui ont été converties en type **Boolean**.
- **Test de la condition.** L'instruction **While** vérifie toujours la condition avant de commencer la boucle. La boucle continue tant que la condition est **True**.
- **Nombre d'itérations.** Si l'élément *condition* est **False** lorsque vous entrez en premier la boucle, il ne s'exécute même pas une seule fois.
- **Imbrication de boucles.** Vous pouvez imbriquer les boucles **While** en plaçant une boucle à l'intérieur d'une autre. Vous pouvez également imbriquer différents types de structures de contrôle les uns dans les autres. Pour plus d'informations, consultez Structures de contrôle imbriquées.
- **Transfert hors de la boucle.** L'instruction Exit, instruction (Visual Basic) passe immédiatement le contrôle à l'instruction qui suit l'instruction **End While**. Vous pouvez éventuellement quitter une boucle si vous décelez une condition qui la rend inutile ou impossible pour poursuivre l'itération, telle qu'une valeur erronée ou une demande d'arrêt. Vous pouvez placer un nombre indéterminé d'instructions **Exit While** n'importe où dans la boucle **While**. **Exit While** est souvent utilisé après évaluation d'une certaine condition, par exemple dans une structure **If...Then...Else**.
- **Boucles infinies.** Vous pouvez utiliser **Exit While** pour tester une condition qui engendrerait une *boucle infinie*, c'est-à-dire une boucle qui pourrait s'exécuter de nombreuses fois, voire indéfiniment. Si vous détectez une telle condition, vous pouvez utiliser **Exit While** pour abandonner la boucle. Pour plus d'informations, consultez Do...Loop, instruction (Visual Basic).

Exemple

Cet exemple utilise la structure **While...End While** pour incrémenter une variable de compteur. Les instructions de la boucle s'exécutent tant que la condition a la valeur **True**.

VB

```
Dim counter As Integer = 0
While counter < 20
    counter += 1
    ' Insert code to use current value of counter.
End While
MsgBox("While loop ran " & CStr(counter) & " times")
```

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)
For...Next, instruction (Visual Basic)
Boolean, type de données (Visual Basic)
Exit, instruction (Visual Basic)

Concepts

Structures de boucle
Structures de contrôle imbriquées

With...End With, instruction (Visual Basic)

Exécute un ensemble d'instructions faisant continuellement référence à un objet ou à une structure unique.

```
With object
    [ statements ]
End With
```

Éléments

object

Obligatoire. Variable ou expression. Peut avoir pour valeur tout type de données, y compris des types élémentaires.

statements

Facultatif. Une ou plusieurs instructions entre **With** et **End With** qui s'exécutent sur *object*.

End With

Obligatoire. Met fin à la définition du bloc **With**.

Notes

With...End With permet d'appliquer une série d'instructions à l'objet spécifié, sans désigner à chaque fois le nom de l'objet. Si le chemin d'accès de qualification à l'objet est long, l'utilisation de **With...End With** peut améliorer vos performances. Un bloc **With** permet également d'éviter les frappes répétées du chemin d'accès de qualification et les erreurs de frappe d'un de ses éléments.

Par exemple, pour modifier plusieurs propriétés d'un seul objet, placez les instructions d'assignation de propriétés dans **With...End With** ; vous ne faites ainsi référence qu'une seule fois à l'objet, au lieu de le faire à chaque assignation de propriété.

Règles

- **Types de données.** Le type de données de *object* peut être un type de classe ou de structure, ou même un type élémentaire Visual Basic, tel que **Integer**. Le .NET Framework prend en charge tous les types élémentaires contenant des classes ou des structures, dont les membres sont accessibles dans un bloc **With**.
- **Déclaration.** Vous devez déclarer *object* avant d'entrer le bloc **With**. Vous ne pouvez pas le déclarer dans l'instruction **With**.
- **Nombre d'itérations.** Un bloc **With** n'est pas une construction itérative. Les instructions ne s'exécutent qu'une seule fois sauf si le bloc contient une boucle.
- **Imbrication des structures.** Vous pouvez imbriquer des structures **With...End With** en plaçant une structure à l'intérieur d'une autre. Pour un exemple, consultez Comment : exécuter plusieurs actions sur un objet.

  Toutefois, les membres des instructions externes étant masqués dans les instructions internes, vous devez fournir dans une instruction **With** interne une référence d'objet complète pour tous les membres d'un objet figurant dans un bloc **With** externe.

  Vous pouvez également imbriquer divers types de structures de contrôle l'un dans l'autre. Pour plus d'informations, consultez Structures de contrôle imbriquées.
- **Transfert en dehors de la structure.** Visual Basic ne prend pas en charge Exit, instruction (Visual Basic) pour transférer le contrôle hors d'un bloc **With**. Si vous souhaitez quitter avant que toutes les instructions ne soient exécutées, insérez une étiquette sur l'instruction **End With** et utilisez l'instruction GoTo pour effectuer un branchement vers cette dernière. Pour plus d'informations, consultez Comment : étiqueter des instructions.

  Vous ne pouvez pas transférer le contrôle de l'extérieur d'un bloc **With** vers l'intérieur de celui-ci, ou de l'intérieur de celui-ci vers l'extérieur. Vous pouvez appeler une procédure de l'intérieur du bloc, mais le contrôle retourne à l'instruction suivante.
- **Accès aux autres objets.** Une fois que vous avez entré un bloc **With**, vous ne pouvez pas réassigner *object* tant que vous n'avez pas passé l'instruction **End With** Vous pouvez dès lors accéder aux méthodes et aux propriétés de l'objet spécifié uniquement sans les qualifier. Vous pouvez utiliser des méthodes et des propriétés pour les autres objets, mais

vous devez les désigner par leurs noms d'objets.

Exemple

L'exemple ci-dessous utilise un bloc **With** pour exécuter une série d'instructions sur un seul objet. Il suppose que l'objet `testObject` a déjà été créé et expose les propriétés référencées.

VB

```
With testObject
    .Height = 100
    .Text = "Hello, World"
    .ForeColor = System.Drawing.Color.Green
    .Font = New System.Drawing.Font(.Font, _
        System.Drawing.FontStyle.Bold)
End With
```

Voir aussi

Tâches

Comment : exécuter plusieurs actions sur un objet

Comment : étiqueter des instructions

Référence

Exit, instruction (Visual Basic)

GoTo, instruction

Concepts

Structures de contrôle imbriquées

Contextes de déclaration et niveaux d'accès par défaut

Cette rubrique décrit quels types Visual Basic il est possible de déclarer dans quels autres types et quelle est la valeur par défaut de leurs niveaux d'accès si ceux-ci ne sont pas spécifiés.

Niveaux de contexte de déclaration

Le *contexte de déclaration* d'un élément de programmation est la région de code dans laquelle il est déclaré. Il s'agit souvent d'un autre élément de programmation, qui est ensuite appelé *élément contenant*.

Les niveaux de contextes de déclaration sont les suivants :

- *Niveau d'espace de noms* — dans un fichier source ou un espace de noms, mais pas dans une classe, une structure, un module ou une interface
- *Niveau de module* — dans une classe, une structure, un module ou une interface, mais pas dans une procédure ou un bloc
- *Niveau de procédure* — dans une procédure ou un bloc (tel que **If** ou **For**)

Le tableau suivant montre les niveaux d'accès par défaut de plusieurs éléments de programmation déclarés, selon leurs contextes de déclaration.

Élément déclaré	Niveau d'espace de noms	Niveau de module	Niveau de procédure
Variable (instruction Dim (Visual Basic))	Non autorisé	**Private** (**Public** dans **Structure**, non autorisé dans **Interface**)	**Public**
Constante (instruction Const (Visual Basic))	Non autorisé	**Private** (**Public** dans **Structure**, non autorisé dans **Interface**)	**Public**
Énumération (instruction Enum (Visual Basic))	**Friend**	**Public**	Non autorisé
Classe (instruction Class (Visual Basic))	**Friend**	**Public**	Non autorisé
Structure (instruction Structure)	**Friend**	**Public**	Non autorisé
Module (instruction Module)	**Friend**	Non autorisé	Non autorisé
Interface (instruction Interface (Visual Basic))	**Friend**	**Public**	Non autorisé
Procédure (instruction Function (Visual Basic), instruction Sub (Visual Basic))	Non autorisé	**Public**	Non autorisé
Référence externe (instruction Declare)	Non autorisé	**Public** (non autorisé dans **Interface**)	Non autorisé
Opérateur (instruction Operator)	Non autorisé	**Public** (non autorisé dans **Interface** ou **Module**)	Non autorisé
Propriété (instruction Property)	Non autorisé	**Public**	Non autorisé
Propriété par défaut (Default (Visual Basic))	Non autorisé	**Public** (non autorisé dans **Module**)	Non autorisé
Événement (instruction Event)	Non autorisé	**Public**	Non autorisé
Délégué (instruction Delegate)	**Friend**	**Public**	Non autorisé

Pour plus d'informations, consultez Niveaux d'accès dans Visual Basic.

Voir aussi

Référence

Friend (Visual Basic)

Private (Visual Basic)

Public (Visual Basic)

Liste d'attributs

Spécifie les attributs à appliquer à un élément de programmation déclaré. Les attributs multiples sont séparés par des virgules. La syntaxe pour un attribut est la suivante.

```
[ attributemodifier ] attributename [ ( attributearguments | attributeinitializer ) ]
```

Éléments

attributemodifier

Requis pour les attributs appliqués au début d'un fichier source. Peut être Assembly ou Module.

attributename

Requis. Nom de l'attribut.

attributearguments

Facultatif. Liste d'arguments de position pour cet attribut. Les arguments multiples sont séparés par des virgules.

attributeinitializer

Facultatif. Liste d'initialiseurs de variable ou de propriété pour cet attribut. Les initialiseurs multiples sont séparés par des virgules.

Notes

Vous pouvez appliquer un ou plusieurs attributs à quasiment n'importe quel élément de programmation (types, procédures, propriétés, etc.). Les attributs apparaissent dans les métadonnées de votre assembly ; ils peuvent vous aider à annoter votre code ou à spécifier comment utiliser un élément de programmation particulier. Vous pouvez appliquer des attributs définis par Visual Basic et le .NET Framework et vous pouvez définir vos propres attributs.

Pour plus d'informations sur le moment où des attributs doivent être utilisés, consultez Attributs en Visual Basic et Utilisations courantes des attributs. Pour plus d'informations sur les noms d'attributs, consultez Noms d'éléments déclarés.

Règles

- **Positionnement.** Vous pouvez appliquer des attributs à la plupart des éléments de programmation déclarés. Pour appliquer un ou plusieurs attributs, placez un *bloc d'attributs* au début de la déclaration d'élément. Chaque entrée dans la liste d'attributs spécifie un attribut que vous souhaitez appliquer, ainsi que le modificateur et les arguments que vous utilisez pour cet appel de l'attribut.
- **Signes "inférieur à" et "supérieur à".** Si vous fournissez une liste d'attributs, vous devez la placer entre des signes "inférieur à" et "supérieur à" ("**<**" et "**>**").
- **Partie de la déclaration.** L'attribut doit faire partie de la déclaration d'élément, et non d'une instruction séparée. Vous pouvez utiliser la séquence de continuation de ligne (" _") pour étendre l'instruction de déclaration sur plusieurs lignes de code source.
- **Modificateurs.** Un modificateur d'attribut (**Assembly** ou **Module**) est requis sur chaque attribut appliqué à un élément de programmation au début d'un fichier source. Les modificateurs d'attribut ne sont pas autorisés sur des attributs appliqués aux éléments qui ne sont pas au début d'un fichier source.
- **Arguments.** Tous les arguments positionnels d'un attribut doivent précéder n'importe quel initialiseur de propriété ou de variable.

Exemple

L'exemple suivant applique l'attribut DllImportAttribute à une définition squelette d'une procédure **Function**.

VB

```
<DllImportAttribute("kernel32.dll", EntryPoint:="MoveFileW", _
    SetLastError:=True, CharSet:=CharSet.Unicode, _
    ExactSpelling:=True, _
    CallingConvention:=CallingConvention.StdCall)> _
```

```
Public Shared Function moveFile(ByVal src As String, _
  ByVal dst As String) As Boolean
    ' This function copies a file from the path src to the path dst.
    ' Leave this function empty. The DLLImport attribute forces calls
    ' to moveFile to be forwarded to MoveFileW in KERNEL32.DLL.
End Function
```

DllImportAttribute indique que la procédure attribuée représente un point d'entrée dans une bibliothèque de liens dynamiques (DLL) non managée. L'attribut fournit le nom de DLL comme argument positionnel et l'autre information comme initialiseurs de variable.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Assembly

Module (Visual Basic)

Concepts

Utilisations courantes des attributs

Application des attributs

Autres ressources

Attributs en Visual Basic

Liste de paramètres

Spécifie les paramètres qu'une procédure attend lorsqu'elle est appelée. Les paramètres, lorsqu'il y en a plusieurs, sont séparés par des virgules. La syntaxe pour un paramètre est la suivante.

```
[ <attributelist> ] [ Optional ] [{ ByVal | ByRef }] [ ParamArray ]
parametername[( )] [ As parametertype ] [ = defaultvalue ]
```

Éléments

attributelist

Facultatif. Liste des attributs qui s'appliquent à ce paramètre. Vous devez placer le Liste d'attributs entre des signes "inférieur à" et "supérieur à" ("<" et ">").

Optional

Facultatif. Spécifie que ce paramètre n'est pas requis lorsque la procédure est appelée.

ByVal

Facultatif. Spécifie que la procédure ne peut pas remplacer ou réassigner l'élément de variable sous-jacent de l'argument correspondant dans le code appelant.

ByRef

Facultatif. Spécifie que la procédure peut modifier l'élément de variable sous-jacent dans le code appelant comme le ferait le code appelant lui-même.

ParamArray

Facultatif. Spécifie que le dernier paramètre dans la liste de paramètres est un tableau facultatif d'éléments du type de données spécifié. Cela permet au code appelant de passer un nombre arbitraire d'arguments à la procédure.

parametername

Requis. Nom de la variable locale représentant le paramètre.

parametertype

Requis si **Option Strict** a la valeur **On**. Type de données de la variable locale représentant le paramètre.

defaultvalue

Requis pour les paramètres **Optional**. Toute constante ou expression constante qui prend la valeur du type de données du paramètre. Si le type est **Object** ou bien une classe, une interface, un tableau ou une structure, la valeur par défaut ne peut être que **Nothing**.

Notes

Les paramètres sont placés entre parenthèses et sont séparés par des virgules. Un paramètre peut être déclaré avec n'importe quel type de données. Si vous ne spécifiez pas *parametertype*, la valeur prise par défaut est **Object**.

Lorsque le code appelant appelle la procédure, il passe un *argument* à chaque paramètre requis. Pour plus d'informations, consultez Différences entre les paramètres et les arguments.

L'argument que le code appelant passe à chaque paramètre est un pointeur vers un élément sous-jacent dans le code appelant. Si cet élément est *non variable* (une constante, un littéral, une énumération ou une expression), aucun code ne pourra le modifier. S'il s'agit d'un élément *variable* (une variable, un champ, une propriété, un élément de tableau ou un élément de structure déclaré), le code appelant peut le modifier. Pour plus d'informations, consultez Différences entre les arguments modifiables et non modifiables.

Si un élément variable est passé **ByRef**, la procédure peut également le modifier. Pour plus d'informations, consultez Différences entre le passage d'un argument par valeur et par référence.

Règles

- **Parenthèses.**Si vous spécifiez une liste de paramètres, vous devez la placer entre parenthèses. S'il n'y a aucun paramètre, vous pouvez tout de même utiliser des parenthèses entourant une liste vide. Cela permet d'améliorer la

lisibilité de votre code en clarifiant que l'élément est une procédure.

- **Paramètres facultatifs.** Si vous utilisez le modificateur **Optional** sur un paramètre, tous les paramètres suivants dans la liste doivent également être facultatifs et être déclarés à l'aide du modificateur **Optional**.

  Chaque déclaration de paramètre facultative doit fournir la clause *defaultvalue*.

- **Tableaux de paramètres.** Vous devez spécifier **ByVal** pour un paramètre **ParamArray**.

  Vous ne pouvez pas utiliser à la fois **Optional** et **ParamArray** dans la même liste de paramètres.

- **Mécanisme de passage.** Le mécanisme par défaut pour chaque argument est **ByVal**, ce qui signifie que la procédure ne peut pas modifier l'élément de variable sous-jacent. Toutefois, si l'élément est un type référence, la procédure peut modifier le contenu ou les membres de l'objet sous-jacent, bien qu'il ne puisse pas remplacer ou réassigner l'objet lui-même.

- **Noms de paramètres.** Si le type de données du paramètre est un tableau, insérez des parenthèses immédiatement après *parametername*. Pour plus d'informations sur les noms de paramètres, consultez Noms d'éléments déclarés.

Exemple

L'exemple suivant indique une procédure **Function** qui définit deux paramètres.

VB

```
Public Function howMany(ByVal ch As Char, ByVal st As String) As Integer
End Function
Dim howManyA As Integer = howMany("a"c, "How many a's in this string?")
```

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Declare, instruction
Structure, instruction
Option Strict, instruction
DllImportAttribute

Concepts

Application des attributs

Autres ressources

Attributs en Visual Basic

Liste de types

Spécifie les *paramètres de type* pour un élément de programmation *générique*. Les paramètres, lorsqu'il y en a plusieurs, sont séparés par des virgules. La syntaxe pour un paramètre de type est la suivante.

```
typename [ As constraintlist ]
```

Éléments

typename

Requis. Nom du paramètre de type. C'est un espace réservé, à remplacer par un type défini fourni par l'argument de type correspondant.

constraintlist

Facultatif. Liste des configurations requises qui contraignent le type de données qui peut être fourni pour *typename*. Si vous avez plusieurs contraintes, placez-les entre accolades (**{ }**) et séparez-les avec des virgules. Vous devez introduire la liste des contraintes avec le mot clé As. Utilisez **As** une seule fois, au début de la liste.

Notes

Chaque élément de programmation générique doit au moins prendre un paramètre de type. Un paramètre de type est un espace réservé pour un type spécifique (un *élément construit*) que le code client spécifie lorsqu'il crée une instance du type générique. Vous pouvez définir une classe, une structure, une interface, une procédure ou un délégué générique.

Pour plus d'informations sur le moment où un type générique doit être défini, consultez Types génériques en Visual Basic. Pour plus d'informations sur les noms de paramètres de type, consultez Noms d'éléments déclarés.

Règles

- **Parenthèses.** Si vous fournissez une liste de paramètres de type, vous devez la placer entre parenthèses, et vous devez introduire la liste avec le mot clé Of. Utilisez **Of** une seule fois, au début de la liste.
- **Contraintes.** Une liste de *contraintes* sur un paramètre de type peut inclure les éléments suivants selon n'importe quelle combinaison :
  - N'importe quel nombre d'interfaces. Le type fourni doit implémenter toutes les interfaces de cette liste.
  - Une classe au plus. Le type fourni doit hériter de cette classe.
  - Le mot clé New (Visual Basic). Le type fourni doit exposer un constructeur sans paramètre auquel votre type générique peut accéder. Cette opération est utile si vous contraignez un paramètre de type par une ou plusieurs interfaces. Un type qui implémente des interfaces n'expose pas nécessairement de constructeur, et selon le niveau d'accès d'un constructeur, il est possible que le code dans le type générique ne puisse pas y accéder.
  - Le mot clé Class (Visual Basic) ou Structure (Visual Basic). Si vous incluez **Class**, le type fourni doit être un type référence. Si vous incluez **Structure**, le type fourni doit être un type valeur. Vous ne pouvez pas inclure à la fois **Class** et **Structure** dans le même *constraintlist*.

  Le type fourni doit satisfaire chaque configuration requise que vous incluez dans *constraintlist*.

  Les contraintes sur chaque paramètre de type sont indépendantes des contraintes sur les autres paramètres de type.

Comportement

- **Substitution au moment de la compilation.** Lorsque vous créez un type construit à partir d'un élément de programmation générique, fournissez un type défini pour chaque paramètre de type. Le compilateur Visual Basic substitue ce type fourni pour chaque occurrence de *typename* dans l'élément générique.
- **Absence de contraintes.** Si vous ne spécifiez pas de contraintes sur un paramètre de type, votre code est limité aux opérations et aux membres pris en charge par le type de données Object pour ce paramètre.

Exemple

L'exemple suivant affiche une définition squelette d'une classe de dictionnaire générique, y compris une fonction squelette

pour ajouter une nouvelle entrée au dictionnaire.

VB

```
Public Class dictionary(Of entryType, keyType As {IComparable, IFormattable, New})
    Public Sub add(ByVal et As entryType, ByVal kt As keyType)
        Dim dk As keyType
        If kt.CompareTo(dk) = 0 Then
        End If
    End Sub
End Class
```

Étant donné que `dictionary` est générique, le code qui l'utilise peut créer divers objets à partir de celui-ci, chacun disposant des mêmes fonctionnalités, mais agissant sur un type de données différent. L'exemple suivant affiche une ligne de code qui crée un objet `dictionary` avec les entrées **String** et les clés **Integer**.

VB

```
Dim dictInt As New dictionary(Of String, Integer)
```

L'exemple suivant affiche la définition squelette équivalente générée par l'exemple précédent.

VB

```
Public Class dictionary
    Public Sub add(ByVal et As String, ByVal kt As Integer)
        Dim dk As Integer
        If kt.CompareTo(dk) = 0 Then
        End If
    End Sub
End Class
```

Voir aussi

Tâches

Comment : utiliser une classe générique

Référence

Of

New (Visual Basic)

Class (Visual Basic)

Structure (Visual Basic)

Object, type de données

Function, instruction (Visual Basic)

Structure, instruction

Sub, instruction (Visual Basic)

Concepts

Niveaux d'accès dans Visual Basic

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Le compilateur Visual Basic peut traiter les commentaires de documentation dans votre code et les place dans un fichier XML. Vous pouvez utiliser des outils supplémentaires pour traiter le fichier XML dans la documentation.

Les commentaires XML sont autorisés sur les constructions de code telles que les types et membres de types. Pour les types partiels, seule une partie du type peut contenir des commentaires XML bien qu'elle ne présente aucune restriction sur le commentaire de ses membres.

Remarque
Les commentaires de documentation ne peuvent pas être appliqués aux espaces de noms. En effet, un seul espace de noms peut couvrir plusieurs assemblys, et il ne faut pas les charger tous en même temps.

Le compilateur traite n'importe quelle balise représentant du XML correct. Les balises suivantes fournissent des fonctionnalités fréquemment employées dans la documentation utilisateur.

<c>	<code>	<example>
<exception> [1]	<include> [1]	<list>
<para>	<param> [1]	<paramref>
<permission> [1]	<remarks>	<returns>
<see> [1]	<seealso> [1]	<summary>
<typeparam> [1]	<value>	

([1] Le compilateur vérifie la syntaxe.)

Remarque
Si vous souhaitez que les signes "inférieur à" et "supérieur à" apparaissent dans le texte d'un commentaire de documentation, utilisez `<` et `>`. Par exemple, la chaîne `"<text in angle brackets>"` apparaît telle que `<text in angle brackets>`.

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Référence

/doc

Concepts

Documentation de votre code avec le langage XML (Visual Basic)

<c> (Visual Basic)

Indique que le texte d'une description constitue du code.

```
<c>text</c>
```

Paramètres
text

Texte que vous souhaitez indiquer en tant que code.

Notes

La balise <c> permet d'indiquer que le texte d'une description doit être marqué comme étant du code. Utilisez <code> (Visual Basic) pour indiquer que plusieurs lignes constituent du code.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise <c> dans la section sommaire pour indiquer que `Counter` constitue du code.

VB

```
''' <summary>
''' Resets the value the <c>Counter</c> field.
''' </summary>
Public Sub ResetCounter()
    counterValue = 0
End Sub
Private counterValue As Integer = 0
''' <summary>
''' Returns the number of times Counter was called.
''' </summary>
''' <value>Number of times Counter was called.</value>
Public ReadOnly Property Counter() As Integer
    Get
        counterValue += 1
        Return counterValue
    End Get
End Property
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<code> (Visual Basic)

Indique que le texte contient plusieurs lignes de code.

```
<code>content</code>
```

Paramètres
content

Texte à marquer comme étant du code.

Notes

Utilisez les balises `<code>` pour indiquer plusieurs lignes comme étant du code. Utilisez <c> (Visual Basic) pour indiquer que le texte d'une description doit être marqué comme étant du code.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise <code> pour inclure l'exemple de code pour utiliser le champ `ID`.

VB

```
Public Class Employee
    ''' <remarks>
    ''' <example> This sample shows how to set the <c>ID</c> field.
    ''' <code>
    ''' Dim alice As New Employee
    ''' alice.ID = 1234
    ''' </code>
    ''' </example>
    ''' </remarks>
    Public ID As Integer
End Class
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<example> (Visual Basic)

Spécifie un exemple pour le membre.

```
<example>description</example>
```

Paramètres
description

Description de l'exemple de code.

Notes

La balise `<example>` vous permet de spécifier un exemple d'utilisation d'une méthode ou de tout autre membre de bibliothèque. Cela implique généralement l'utilisation de la balise <code> (Visual Basic).

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<example>` pour inclure un exemple afin d'utiliser le champ `ID`.

VB

```
Public Class Employee
    ''' <remarks>
    ''' <example> This sample shows how to set the <c>ID</c> field.
    ''' <code>
    ''' Dim alice As New Employee
    ''' alice.ID = 1234
    ''' </code>
    ''' </example>
    ''' </remarks>
    Public ID As Integer
End Class
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<exception> (Visual Basic)

Spécifie les exceptions qui peuvent être levées.

```
<exception cref="member">description</exception>
```

Paramètres
member

Référence à une exception disponible dans l'environnement de compilation en cours. Le compilateur vérifie que l'exception donnée existe et traduit *member* par le nom d'élément complet dans la sortie XML. *member* doit être placé entre guillemets doubles (" ").

description

Description.

Notes

Utilisez les balises `<exception>` pour spécifier quelles exceptions peuvent être levées. Cette balise est appliquée à une définition de méthode.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise les balises `<exception>` pour décrire une exception que la fonction `IntDivide` peut lever.

VB

```
''' <exception cref="System.OverflowException">
''' Thrown when <paramref name="denominator"/><c> = 0</c>.
''' </exception>
Public Function IntDivide( _
        ByVal numerator As Integer, _
        ByVal denominator As Integer _
) As Integer
    Return numerator \ denominator
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<include> (Visual Basic)

Fait référence à un autre fichier qui décrit les types et membres dans votre code source.

```
<include file="filename" path="tagpath[@name='id']" />
```

Paramètres

filename

Requis. Nom du fichier contenant la documentation. Le nom du fichier peut être précisé par un chemin. Mettez *filename* entre guillemets doubles (" ").

tagpath

Requis. Chemin d'accès des balises dans *filename* qui permet d'accéder au *name* de la balise. Mettez le chemin d'accès entre guillemets doubles (" ").

name

Requis. Dans la balise, spécificateur de nom précédant les commentaires. *Name* est associé à un *id*.

id

Requis. ID de la balise qui précède les commentaires. Mettez l'ID entre guillemets simples (' ').

Notes

Utilisez la balise `<include>` pour faire référence à des commentaires situés dans un autre fichier qui décrivent les types et membres dans votre code source. Cette solution remplace celle consistant à placer des commentaires de documentation directement dans votre fichier de code source.

La balise `<include>` utilise la recommandation du W3C intitulée XML Path Language (XPath) Version 1.0. Plus d'informations sur les moyens de personnaliser votre utilisation de `<include>` sont disponibles à la page http://www.w3.org/TR/xpath.

Exemple

Cet exemple utilise la balise `<include>` pour importer des commentaires de documentation membre d'un fichier nommé `commentFile.xml`.

VB

```
''' <include file="commentFile.xml"
''' path="Docs/Members[@name='Open']/*" />
Public Sub Open(ByVal filename As String)
    ' Code goes here.
End Sub
''' <include file="commentFile.xml"
''' path="Docs/Members[@name='Close']/*" />
Public Sub Close(ByVal filename As String)
    ' Code goes here.
End Sub
```

Le format du `commentFile.xml` est le suivant :

```
<Docs>
<Members name="Open">
<summary>Opens a file.</summary>
<param name="filename">File name to open.</param>
</Members>
<Members name="Close">
<summary>Closes a file.</summary>
<param name="filename">File name to close.</param>
</Members>
</Docs>
```

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<list> (Visual Basic)

Définit une liste ou une table.

```
<list type="type">
   <listheader>
      <term>term</term>
      <description>description</description>
   </listheader>
   <item>
      <term>term</term>
      <description>description</description>
   </item>
</list>
```

Paramètres

type

Type de la liste. Doit être une "puce" pour une liste à puces, "numéro" pour une liste numérotée, ou "table" pour une table à deux colonnes.

term

Utilisé uniquement lorsque *type* est "table." Terme à définir qui est défini dans la balise de description.

description

Lorsque *type* est "puce" ou "numéro", *description* est un élément dans la liste. Lorsque *type* est "table", *description* est la définition de *term*.

Notes

Le bloc `<listheader>` permet de définir le titre d'une table ou d'une liste de définitions. Lorsque vous définissez une table, il suffit de fournir une entrée pour *term* dans le titre.

Chaque élément de la liste est spécifié dans un bloc `<item>`. Lorsque vous créez une liste de définitions, vous devez spécifier à la fois *term* et *description*. Cependant, pour une table, une liste à puces ou une liste numérotée, il vous suffit de fournir une entrée pour *description*.

Une liste ou une table peut comporter autant de blocs `<item>` que nécessaire.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<list>` pour définir une liste à puces dans la section Notes.

VB

```
''' <remarks>Before calling the <c>Reset</c> method, be sure to:
''' <list type="bullet">
''' <item><description>Close all connections.</description></item>
''' <item><description>Save the object state.</description></item>
''' </list>
''' </remarks>
Public Sub Reset()
    ' Code goes here.
End Sub
```

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<para> (Visual Basic)

Spécifie que le contenu est mis en forme comme un paragraphe.

```
<para>content</para>
```

Paramètres
content

Texte du paragraphe.

Notes

La balise `<para>` est à utiliser à l'intérieur d'une balise, telle que <summary> (Visual Basic), <remarks> (Visual Basic) ou <returns> (Visual Basic) et vous permet d'ajouter une structure au texte.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<para>` pour fractionner la section Notes de la méthode `UpdateRecord` en deux paragraphes.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<param> (Visual Basic)

Définit un nom et une description de paramètre.

```
<param name="name">description</param>
```

Paramètres

name

Nom d'un paramètre de méthode. Mettez le nom entre guillemets doubles (" ").

description

Description du paramètre.

Notes

La balise `<param>` doit être utilisée dans le commentaire pour une déclaration de méthode afin de décrire l'un des paramètres pour la méthode.

Le texte de la balise `<param>` sera affiché dans les Options IntelliSense et dans l'Explorateur d'objets.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<param>` pour décrire le paramètre `id`.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<paramref> (Visual Basic)

Met en forme un mot comme un paramètre.

```
<paramref name="name"/>
```

Paramètres
name

Nom du paramètre de référence. Mettez le nom entre guillemets doubles (" ").

Notes

La balise `<paramref>` vous permet d'indiquer qu'un mot est un paramètre. Le fichier XML peut être traité de manière à ce que ce paramètre ait un format bien distinct.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<paramref>` pour faire référence au paramètre `id`.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<permission> (Visual Basic)

Spécifie une autorisation requise pour le membre.

```
<permission cref="member">description</permission>
```

Paramètres

member

Référence à un membre ou un champ qu'il est possible d'appeler depuis l'environnement de compilation actuel. Le compilateur vérifie que l'élément de code donné existe et traduit *member* dans le nom d'élément canonique dans le fichier de sortie XML. Mettez *member* entre guillemets (" ").

description

Description de l'accès au membre.

Notes

Utilisez la balise `<permission>` pour documenter l'accès d'un membre. Utilisez la classe PermissionSet pour spécifier l'accès à un membre.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<permission>` pour décrire que le FileIOPermission est requis par la méthode `ReadFile`.

VB

```
''' <permission cref="System.Security.Permissions.FileIOPermission">
''' Needs full access to the specified file.
''' </permission>
Public Sub ReadFile(ByVal filename As String)
    ' Code goes here.
End Sub
```

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<remarks> (Visual Basic)

Spécifie une section Notes pour le membre.

```
<remarks>description</remarks>
```

Paramètres
description

Description du membre.

Notes

La balise `<remarks>` permet d'ajouter des informations sur un type et de compléter les informations spécifiées par <summary> (Visual Basic). Ces informations sont affichées dans l'Explorateur d'objets.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<remarks>` pour expliquer le rôle de la méthode `UpdateRecord` retourne.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<returns> (Visual Basic)

Spécifie la valeur de retour de la propriété ou de la fonction.

```
<returns>description</returns>
```

Paramètres
description

Description de la valeur retournée.

Notes

Utilisez la balise `<returns>` dans le commentaire pour une déclaration de méthode afin de décrire la valeur de retour.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<returns>` pour expliquer ce que la fonction `DoesRecordExist` retourne.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<see> (Visual Basic)

Spécifie un lien vers un autre membre.

```
<see cref="member"/>
```

Paramètres
member

Référence à un membre ou un champ qu'il est possible d'appeler depuis l'environnement de compilation actuel. Le compilateur vérifie que l'élément de code donné existe et passe *member* au nom d'élément dans la sortie XML. *member* doit être placé entre guillemets doubles (" ").

Notes

Utilisez la balise `<see>` pour spécifier un lien à partir de l'intérieur du texte. Utilisez <seealso> (Visual Basic) pour indiquer du texte que vous souhaitez voir apparaître dans une section « Voir aussi ».

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise les balises `<see>` dans la section Notes `UpdateRecord` pour faire référence à la méthode `DoesRecordExist`.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<seealso> (Visual Basic)

Spécifie un lien qui apparaît à la section Voir aussi.

```
<seealso cref="member"/>
```

Paramètres
member

Référence à un membre ou un champ qu'il est possible d'appeler depuis l'environnement de compilation actuel. Le compilateur vérifie que l'élément de code donné existe et passe *member* au nom d'élément dans la sortie XML. *member* doit être placé entre guillemets doubles (" ").

Notes

Utilisez les balises `<seealso>` pour spécifier le texte que vous souhaitez voir apparaître à la section Voir aussi. Utilisez <see> (Visual Basic) pour spécifier un lien à partir de l'intérieur du texte.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise les balises `<seealso>` dans la section Notes `DoesRecordExist` pour faire référence à la méthode `UpdateRecord`.

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<summary> (Visual Basic)

Spécifie le résumé du membre.

```
<summary>description</summary>
```

Paramètres
description

Résumé de l'objet.

Notes

Utilisez les balises `<summary>` pour décrire un type ou un membre de type. Utilisez <remarks> (Visual Basic) pour ajouter des informations complémentaires à une description de type.

Le texte de la balise `<summary>` est la seule source d'informations sur le type dans la zone IntelliSense et est également affiché dans l'Explorateur d'objets.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<summary>` pour décrire la méthode `ResetCounter` et la propriété `Counter`.

VB

```
''' <summary>
''' Resets the value the <c>Counter</c> field.
''' </summary>
Public Sub ResetCounter()
    counterValue = 0
End Sub
Private counterValue As Integer = 0
''' <summary>
''' Returns the number of times Counter was called.
''' </summary>
''' <value>Number of times Counter was called.</value>
Public ReadOnly Property Counter() As Integer
    Get
        counterValue += 1
        Return counterValue
    End Get
End Property
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<typeparam> (Visual Basic)

Définit un nom de paramètre de type et une description.

```
<typeparam name="name">description</typeparam>
```

Paramètres

name

Nom du paramètre de type. Mettez le nom entre guillemets doubles (" ").

description

Description du paramètre de type.

Notes

Utilisez la balise `<typeparam>` dans le commentaire d'un type générique ou une déclaration de membre générique pour décrire l'un des paramètres de type.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<typeparam>` pour décrire le paramètre `id`.

VB

```
''' <typeparam name="T">
''' The base item type. Must implement IComparable.
''' </typeparam>
Public Class itemManager(Of T As IComparable)
    ' Insert code that defines class members.
End Class
```

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

<value> (Visual Basic)

Spécifie la description d'une propriété.

```
<value>property-description</value>
```

Paramètres
property-description

Description de la propriété.

Notes

Utilisez la balise `<value>` pour décrire une propriété. Remarquez que lorsque vous ajoutez une propriété en utilisant l'Assistant Code dans l'environnement de développement Visual Studio .NET, une balise <summary> (Visual Basic) est ajoutée pour la nouvelle propriété. Vous devez ensuite ajouter manuellement une balise `<value>` pour décrire la valeur représentée par la propriété.

Compilez avec /doc pour traiter les commentaires de documentation et les placer dans un fichier.

Exemple

Cet exemple utilise la balise `<value>` pour décrire la valeur que contient la propriété `Counter`.

VB

```
''' <summary>
''' Resets the value the <c>Counter</c> field.
''' </summary>
Public Sub ResetCounter()
    counterValue = 0
End Sub
Private counterValue As Integer = 0
''' <summary>
''' Returns the number of times Counter was called.
''' </summary>
''' <value>Number of times Counter was called.</value>
Public ReadOnly Property Counter() As Integer
    Get
        counterValue += 1
        Return counterValue
    End Get
End Property
```

Voir aussi
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Messages d'erreur (Visual Basic)

Des messages d'erreur peuvent être générés pendant l'exécution d'une application, soit dans l'environnement Visual Basic, soit en tant que fichier exécutable autonome. Certains d'entre eux peuvent également s'afficher au moment du design ou de la compilation. À l'aide de l'instruction **On Error** et de la propriété **Number** de l'objet **Err**, vous pouvez provoquer des erreurs récupérables pour les tester et résoudre les problèmes associés. Les numéros d'erreur non utilisés dans la plage 1 à 1 000 sont réservés à une utilisation ultérieure par Visual Basic.

Remarque

La description "Erreur définie par l'application ou par l'objet" s'affiche lorsqu'une erreur est générée à l'aide de la méthode **Raise** de l'objet **Err** ou de l'instruction **Error**, mais que le numéro ne correspond pas à une erreur définie par le langage Visual Basic. Des erreurs de ce type peuvent être définies par l'application hôte (par exemple, Microsoft Excel ou Visual Basic). Cependant, si vous souhaitez les générer à partir du code, vous devez utiliser la méthode **Raise** et renseigner tous les arguments pertinents.

Lorsque vous développez des applications en Visual Basic, vous pouvez obtenir des informations sur un message en appuyant sur F1 pour accéder à l'aide. De même, vous pouvez appuyer sur F1 pour obtenir des informations sur les messages d'erreur affichés dans la fenêtre **Liste des tâches**.

Dans cette section

Comment : obtenir des informations sur les erreurs du compilateur en Visual Basic

Explique le concept des erreurs du compilateur et comment obtenir de l'aide lorsque des messages d'erreur particuliers s'affichent.

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic

Explique le concept des erreurs du compilateur et comment obtenir de l'aide lorsque des messages d'erreur particuliers s'affichent.

Messages du compilateur Visual Basic

Contient les messages d'erreur Visual Basic qui surviennent au moment de la compilation.

Messages d'exécution Visual Basic

Contient les messages d'erreur Visual Basic qui surviennent au moment de l'exécution

Rubriques connexes

Gestion des exceptions et des erreurs en Visual Basic

Répertorie les liens vers les rubriques liées à la gestion des exceptions en Visual Basic.

Procédure pas à pas : gestion structurée des exceptions

Illustre l'utilisation de l'instruction **Try...Catch...Finally** pour gérer des exceptions.

Comment : obtenir des informations sur les erreurs du compilateur en Visual Basic

Les erreurs de compilation Visual Basic se produisent lorsque le compilateur rencontre des problèmes dans le code. Le code responsable des erreurs est souligné d'un trait ondulé dans l'éditeur de code et, lorsque la souris s'arrête juste au-dessus de ce code ainsi marqué, le message d'erreur s'affiche. Les messages s'affichent également dans la **Liste des tâches**.

Obtention d'aide

La fonctionnalité **Correction automatique de la compilation intelligente** suggère des corrections pour certaines erreurs et vous permet d'appliquer le correctif. Lorsqu'une erreur se produit, si un symbole rectangulaire creux apparaît sous le côté droit de la ligne ondulée, vous pouvez le convertir en un panneau des balises actives en plaçant le pointeur de votre souris sur la ligne ondulée ou en double-cliquant sur le message d'erreur qui s'affiche dans la **Liste des tâches**. Lorsque vous cliquez ou placez le pointeur de votre souris sur le panneau des balises actives, la fenêtre du programme d'assistance Options de correction d'erreurs s'ouvre et affiche une description de l'erreur, ainsi que des suggestions pour la corriger. Pour plus d'informations, consultez Comment : corriger les erreurs de compilateur à l'aide de la correction automatique.

Le compilateur Visual Basic contient un ensemble d'avertissements sur le code qui peut générer des erreurs d'exécution. Les développeurs peuvent utiliser ces informations pour écrire du code plus précis et rapide qui crée moins de bogues. Par exemple, le compilateur émet un avertissement lorsque l'utilisateur essaie d'appeler un membre d'une variable objet non assignée, de retourner une fonction sans définir la valeur de retour ou d'exécuter un bloc **Try** contenant des erreurs dans la logique d'interception des exceptions. Pour en savoir plus sur les avertissements, notamment leur activation et désactivation, consultez Configuration d'avertissements en Visual Basic.

Pour obtenir de l'aide sur un message d'erreur donné

1. Mettez en surbrillance le message d'erreur dans la **Liste des tâches**, puis appuyez sur F1.

   - ou -

2. Dans le menu **?** (Aide), cliquez sur **Rechercher**.

3. Tapez le message d'erreur dans la zone **Rechercher** de la fenêtre **Rechercher**.

Remarque
N'oubliez pas que les messages d'erreur affichés contiennent souvent des identificateurs provenant du code, par exemple des noms de variables ou de projets, ainsi que les noms de types de ces éléments, qui ne font pas partie du message d'erreur sous-jacent. Veillez à ne pas utiliser ces mots dans votre recherche.

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic
Comment : déboguer le code dans l'Éditeur
Comment : corriger les erreurs de compilateur à l'aide de la correction automatique

Concepts

Configuration d'avertissements en Visual Basic

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic

Les erreurs d'exécution Visual Basic se produisent lorsqu'une application tente d'effectuer une action que le système ne peut pas exécuter.

Les erreurs levées par Visual Basic sont des objets **Exception**. Visual Basic peut générer des erreurs personnalisées de tout type de données, y compris des objets **Exception**, en utilisant l'instruction **Throw**. Un programme peut afficher le numéro et le message d'erreur d'une exception interceptée pour identifier l'erreur. Si une erreur n'est pas interceptée, le programme se termine.

Les erreurs d'exécution peuvent être interceptées et examinées par le code. En plaçant le code qui génère l'erreur dans un bloc **Try**, toute erreur levée peut être interceptée avec un bloc **Catch** correspondant.

Obtention d'aide

Pour obtenir de l'aide sur un message d'erreur donné

1. Dans le menu **?** (Aide), cliquez sur **Rechercher**.
2. Tapez le message d'erreur dans la zone **Rechercher** de la fenêtre **Rechercher**.

Remarque
Certains messages d'erreur contiennent des mots entre guillemets simples qui font référence à des identificateurs prov enant du code, par exemple un nom de projet ou de variable, et qui ne font pas partie du message d'erreur. N'incluez p as les mots placés entre guillemets dans votre recherche.

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs du compilateur en Visual Basic
Comment : corriger des erreurs d'exécution avec l'Assistant Exception

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Erreurs récupérables dans Visual Basic

Alors que le modèle de gestion d'exceptions structuré fourni dans Visual Basic est plus utile pour développer des applications, vous pouvez choisir d'employer la gestion d'erreurs non structurée, en particulier lors de la mise à niveau d'applications héritées. Vous pouvez utiliser l'On Error, instruction (Visual Basic) pour intercepter des erreurs spécifiques et y répondre.

Erreurs récupérables

Les erreurs récupérables peuvent se produire pendant qu'une application s'exécute. Certaines erreurs récupérables peuvent également se produire pendant le développement ou la compilation. À l'aide de l'instruction **On Error** et de l'objet Err, objet (Visual Basic), vous pouvez tester et résoudre les erreurs récupérables.

Le tableau suivant répertorie les codes d'erreur et leurs messages associés.

Code	Message
0	Pas d'erreur
3	Retourne sans **GoSub** (obsolète, plus utilisé)
5	Argument ou appel de procédure non valide (Visual Basic)
6	Dépassement de capacité (erreur Visual Basic) Dépassement de capacité (erreur d'exécution Visual Basic)
7	Mémoire insuffisante (erreur du compilateur Visual Basic) Mémoire insuffisante (Erreur d'exécution Visual Basic)
9	Indice hors limites (Visual Basic)
10	Ce tableau est prédéfini ou est temporairement verrouillé (Visual Basic)
11	Division par zéro (erreur d'exécution Visual Basic)
13	Incompatibilité de type (Visual Basic)
14	Espace de chaîne insuffisant (Visual Basic)
16	Expression trop complexe
17	Impossible d'effectuer l'opération demandée (Visual Basic)
18	Interruption par l'utilisateur
20	Reprendre sans gestion d'erreur
28	Espace de pile insuffisant (Visual Basic)
35	Sub ou Function non défini (Visual Basic)
47	Trop de clients pour cette DLL

48	Erreur de chargement de la DLL (Visual Basic)
49	Convention d'appel de DLL incorrecte
51	Erreur interne (Visual Basic)
52	Nom ou numéro de fichier incorrect
53	Fichier introuvable (erreur d'exécution Visual Basic)
54	Mode de fichier incorrect
55	Le fichier est déjà ouvert.
57	Erreur d'E/S de périphérique
58	Fichier déjà existant
59	Longueur d'enregistrement incorrecte
61	Disque plein (Visual Basic)
62	L'entrée dépasse la fin du fichier
63	Numéro d'enregistrement incorrect
67	Trop de fichiers
68	Périphérique non disponible
70	Autorisation refusée (Visual Basic)
71	Disque non prêt
74	Impossible de renommer avec un lecteur différent
75	Erreur dans le chemin d'accès
76	Chemin d'accès introuvable
91	Variable objet ou variable bloc With non définie
92	Boucle For non initialisée (Visual Basic)

93	Chaîne de masque non valide
94	Utilisation non valide de **Null** (obsolète, plus utilisé)
95	Erreur définie par l'application ou par l'objet
96	Impossible de recevoir des événements de l'objet, car le nombre maximal de récepteurs d'événements pris en charge est déjà atteint
97	Impossible d'appeler une fonction Friend pour un objet qui n'est pas une instance de la classe de définition
98	Un appel à une propriété ou une méthode ne peut pas utiliser une référence vers un objet privé, que ce soit comme argument ou comme valeur de retour
1000	La classe '<nomclasse>' n'implémente pas l'interface System.Collections.ICollection
298	Impossible de charger la ressource système ou la DLL
320	Impossible d'utiliser des noms de périphériques dans des noms de fichiers spécifiés
321	Le format de fichier n'est pas valide
322	Impossible de créer le fichier temporaire nécessaire
325	Le format n'est pas valide dans le fichier de ressources
327	La valeur des données nommée est introuvable
328	Paramètre non valide ; impossible d'écrire des tableaux
335	Impossible d'accéder au Registre système
336	Le composant ActiveX n'est pas correctement enregistré
337	Composant ActiveX introuvable
338	Le composant ActiveX ne s'est pas exécuté correctement
360	Cet objet est déjà chargé

361	Impossible de charger ou de décharger cet objet
363	Le contrôle ActiveX spécifié est introuvable
364	Cet objet a été déchargé
365	Impossible de décharger dans ce contexte
368	Le fichier spécifié est obsolète
371	L'objet spécifié ne peut pas être utilisé comme formulaire propriétaire pour **Show**
380	Valeur de propriété non valide
381	L'index de tableau de propriétés est non valide
382	Set non pris en charge au moment de l'exécution
383	Set non pris en charge (propriété en lecture seule)
385	Index de tableau de propriétés requis
387	Set non autorisé
393	Get non pris en charge au moment de l'exécution
394	Get non pris en charge (propriété en écriture seule)
400	Formulaire déjà affiché ; impossible de l'afficher sous forme modale
402	Le code doit d'abord fermer le formulaire modal de premier plan

422	Propriété introuvable
423	Propriété ou méthode introuvable
424	Objet requis (Visual Basic)
429	Impossible de créer le composant ActiveX
430	La classe ne prend pas en charge Automation ou l'interface attendue
432	Nom du fichier ou de la classe introuvable lors de l'opération Automation (Visual Basic)
438	L'objet ne prend pas en charge cette propriété ou méthode (Visual Basic)
440	Erreur Automation
442	La connexion à la bibliothèque de types ou d'objets pour processus distant a été perdue
443	L'objet Automation n'a pas de valeur par défaut
445	L'objet ne prend pas en charge cette action (Visual Basic)
446	Cet objet ne prend pas en charge les arguments nommés
447	L'objet ne prend pas en charge les paramètres régionaux actuels (Visual Basic)
448	Argument nommé introuvable (Visual Basic)
449	Argument non facultatif (Visual Basic)
450	Nombre incorrect d'arguments ou assignation de propriété non valide
451	La procédure Property Let n'est pas définie et la procédure Property Get n'a pas retourné d'objet

452	Nombre non valide
453	Fonction DLL spécifiée introuvable
454	Ressource de code introuvable
455	Erreur de verrouillage de ressource de code
457	Cette clé est déjà associée à un élément de cette collection
458	Cette variable utilise un type Automation non pris en charge par Visual Basic
459	L'objet ou la classe ne prend pas en charge le jeu d'événements
460	Format de Presse-papiers non valide
461	Méthode ou données membres introuvables
462	Le serveur distant n'existe pas ou n'est pas disponible (Visual Basic)
463	La classe n'est pas inscrite sur l'ordinateur local
481	Caractère non valide
482	Erreur de l'imprimante
735	Impossible d'enregistrer le fichier dans TEMP
744	Le texte recherché est introuvable
746	Remplacements trop longs
999	Instruction Stop rencontrée

32768	Cette fonctionnalité n'est pas encore implémentée

Le message Erreur définie par l'application ou par l'objet s'applique à un code d'erreur que Visual Basic ne reconnaît pas.

Voir aussi

Tâches

Résolution des problèmes liés à la gestion des exceptions

Référence

On Error, instruction (Visual Basic)

Err, objet (Visual Basic)

Concepts

Introduction à la gestion des exceptions

Vue d'ensemble de la gestion non structurée des exceptions

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Autres ressources

Gestion structurée des exceptions en Visual Basic

Aucune aide n'est disponible pour cette erreur Visual Basic

No Help available for this Visual Basic error

La plupart des messages d'erreur ont des pages d'aide associées. Toutefois, votre projet ou votre code a généré une erreur pour laquelle aucune page d'aide n'existe.

Pour corriger cette erreur

1. Si l'aide en ligne n'est pas activée, activez-la et vérifiez la présence d'une éventuelle documentation de cette erreur. Reportez-vous à Sources de l'aide : aide en local et aide en ligne.
2. Recherchez une éventuelle documentation de cette erreur dans la Base de connaissances (KB) Microsoft (http://support.microsoft.com/search/).
3. Consultez la bibliothèque MSDNhttp://msdn.microsoft.com/library/default.asp) pour rechercher une éventuelle documentation de cette erreur.
4. Si vous ne trouvez aucune documentation, rassemblez des informations à propos des circonstances et contactez les Services de support technique de Microsoft.

   Un renseignement qui pourrait être utile est l'ID d'erreur. Il se compose des lettres BC suivies d'un code à 5 chiffres, par exemple BC30648. Vous pouvez le trouver dans la fenêtre **Sortie**. Pour afficher la fenêtre **Sortie**, sélectionnez **Autres fenêtres** dans le menu **Affichage**, puis choisissez **Sortie**. Vous pouvez également appuyer sur CTRL+ALT+O. Pensez à ces points :

   - Vous devez exécuter le compilateur pour générer l'erreur dans la fenêtre **Sortie**. Pour cela, appuyez sur F5 ou sélectionnez **Démarrer le débogage** dans le menu **Déboguer**.
   - À l'intérieur de la fenêtre **Sortie**, pour « Afficher la sortie à partir de : » sélectionnez **Générer**.

Voir aussi

Tâches

Recherche des fichiers Readme

Référence

Aide en ligne, Environnement, boîte de dialogue Options

Concepts

Sources de l'aide : aide en local et aide en ligne

Aide auprès des Services du Support Technique Microsoft (Visual Studio)*

Autres ressources

Utilisation de l'aide de Visual Studio

Support technique et accessibilité

Messages du compilateur Visual Basic

Cette section contient les messages d'erreur Visual Basic qui se produisent au moment de la compilation.

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs du compilateur en Visual Basic

Autres ressources

Messages d'erreur (Visual Basic)

'>' attendu

'>' expected

La syntaxe du code spécifié est incorrecte. Cette erreur indique souvent que le crochet pointu fermant d'un attribut est manquant.

ID d'erreur : BC30636

Pour corriger cette erreur

- Vérifiez la syntaxe du code utilisé.

Voir aussi

Concepts

Instructions de déclaration en Visual Basic

Autres ressources

Attributs en Visual Basic

'=' attendu (assignation Let ou Set)

'=' expected (Let or Set assignment)

Une instruction d'assignation **Let** ou **Set** ne comprend pas de signe égal (**=**).

Les instructions d'assignation **Let** et **Set** ne sont pas prises en charge dans Visual Basic 2005. Le compilateur peut déterminer le type d'assignation à partir du type de données de l'élément de programmation qui doit recevoir la valeur. Une assignation de *type valeur* remplace l'instruction **Let**, tandis qu'une assignation de *type référence* remplace l'ancienne instruction **Set**.

La nouvelle Set, instruction (Visual Basic) est utilisée dans les définitions de propriété et n'a aucune relation avec l'instruction d'assignation **Set**.

ID d'erreur : BC32020

Pour corriger cette erreur

- Utilisez une instruction d'assignation standard, contenant un signe égal (**=**) et une valeur à assigner à la variable ou à la propriété.

Voir aussi

Concepts

Récapitulatif des modifications relatives à la prise en charge des éléments de programmation
Types valeur et types référence
Instructions d'assignation

'=' attendu (déclaration)

'=' expected (Declaration)

Une instruction **Mid** ou une directive de compilation **#Const** ne comprend pas d'assignation d'une valeur.

ID d'erreur : BC30249

Pour corriger cette erreur

- Ajoutez un signe égal (**=**) et une valeur à assigner à la chaîne ou à la constante de compilation.

Voir aussi

Référence

Mid, instruction

#Const, directive

Concepts

Instructions de déclaration en Visual Basic

Instructions d'assignation

'<nomvariable>' ne peut pas être utilisé en tant que variable 'Catch', car ce n'est ni une variable locale ni un paramètre

'<variablename>' is not a local variable or parameter, and so cannot be used as a 'Catch' variable

Les variables dans les instructions **Try...Catch...Finally** doivent être déclarées localement ou être des paramètres pour la procédure en cours.

ID d'erreur : BC31082

Pour corriger cette erreur

- Déclarez des variables locales ou des paramètres pour les instructions **Try...Catch...Finally**.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Impossible de convertir '<nomtype1>' en '<nomtype2>'

'<typename1>' cannot be converted to '<typename2>'

Une tentative de conversion de type a été effectuée entre deux types incompatibles.

ID d'erreur : BC30741

Pour corriger cette erreur

- Vérifiez les types sur lesquels un cast a été effectué.

Voir aussi

Référence

CType, fonction

Impossible de convertir les valeurs '<nomtype>' en 'Char'

Message d'erreur
Impossible de convertir les valeurs '<nomtype>' en 'Char'. Utilisez Microsoft.VisualBasic.ChrW pour interpréter une valeur numérique en tant que caractère Unicode ou convertissez-la d'abord en 'String' pour obtenir un chiffre.
'<typename>' values cannot be converted to Char. Use Microsoft.VisualBasic.ChrW to interpret a numeric value as a Unicode character or first convert it to 'String' to produce a digit.

Une expression tente de convertir un type de données autre que **String** ou **Object** en **Char**.

ID d'erreur : BC32007

Pour corriger cette erreur

- Utilisez la fonction **ChrW** pour convertir une valeur numérique en caractère Unicode ou convertissez d'abord la valeur en **String** puis en **Char**.

Voir aussi

Référence
Fonctions Chr, ChrW
Char, type de données (Visual Basic)
Concepts
Conversions implicites et explicites

'<nomtype>' doit être déclaré 'MustInherit', car il contient des méthodes déclarées 'MustOverride'

'<typename>' must be declared 'MustInherit' because it contains methods declared 'MustOverride'

Les types qui contiennent des membres **MustOverride** doivent être déclarés **MustInherit**.

ID d'erreur : BC31411

Pour corriger cette erreur

- Ajoutez le modificateur **MustInherit** au type.

Voir aussi

Référence

MustInherit
MustOverride

'<nomtype>' est un type non pris en charge

'<typename>' is an unsupported type

Le type spécifié ne peut pas être représenté par Visual Studio.

ID d'erreur : BC30649

Pour corriger cette erreur

- Utilisez un type pris en charge.
- Développez un nouveau type.

Voir aussi

Référence

Liste des types de données (Visual Basic)

'<nomtype>' est un type interface et ne peut pas être utilisé en tant qu'expression

'<typename>' is an interface type and cannot be used as an expression

Un nom de type apparaît lorsqu'une expression est requise. Une expression doit être constituée d'une combinaison de variables, de constantes, de littéraux, de propriétés et d'appels de procédure **Function**.

ID d'erreur : BC30111

Pour corriger cette erreur

- Supprimez le nom de type et construisez l'expression à l'aide d'éléments valides.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

'<nomtype>' est un type Enum et ne peut pas être utilisé en tant qu'expression

'<typename>' is an Enum type and cannot be used as an expression

Un nom de type apparaît lorsqu'une expression est requise. Une expression doit être constituée d'une combinaison de variables, de constantes, de littéraux, de propriétés et d'appels de procédure **Function**.

ID d'erreur : BC30107

Pour corriger cette erreur

- Supprimez le nom de type et construisez l'expression à l'aide d'éléments valides.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

'<nomtype>' est un type dans '<nomconteneur>' et ne peut pas être utilisé en tant qu'expression

'<typename>' is a type in '<containername>' and cannot be used as an expression

L'un des noms utilisés est déjà déclaré en tant que nom d'un type et ne peut pas être utilisé dans une expression.

ID d'erreur : BC30691

Pour corriger cette erreur

- Attribuez un nom unique au membre.
- Mettez le nom du membre entre crochets.

Voir aussi

Concepts

Utilisation des mots clés comme noms d'éléments dans le code
Conventions d'affectation de noms Visual Basic

'<nomtype>' est un type et ne peut pas être utilisé en tant qu'expression

'<typename>' is a type and cannot be used as an expression

Un nom de type apparaît lorsqu'une expression est requise. Une expression doit être constituée d'une combinaison de variables, de constantes, de littéraux, de propriétés et d'appels de procédure **Function**.

ID d'erreur : BC30108

Pour corriger cette erreur

- Supprimez le nom de type et construisez l'expression à l'aide d'éléments valides.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

'<nomtype>' est un type structure et ne peut pas être utilisé en tant qu'expression

'<typename>' is a structure type and cannot be used as an expression

Un nom de type apparaît lorsqu'une expression est requise. Une expression doit être constituée d'une combinaison de variables, de constantes, de littéraux, de propriétés et d'appels de procédure **Function**.

ID d'erreur : BC30110

Pour corriger cette erreur

- Supprimez le nom de type et construisez l'expression à l'aide d'éléments valides.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

'<nomtype>' est un espace de noms et ne peut pas être utilisé en tant qu'expression

'<typename>' is a namespace and cannot be used as an expression

Un nom de type apparaît lorsqu'une expression est requise. Une expression doit être constituée de variables, constantes, littéraux, propriétés et appels de procédure **Function**.

ID d'erreur : BC30112

Pour corriger cette erreur

- Supprimez le nom de type et construisez l'expression à l'aide d'éléments valides.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

'<nomtype>' est un type délégué

Message d'erreur
'<nomtype>' est un type délégué. Une construction déléguée accepte une seule expression AddressOf en tant que liste d'arguments. Une expression AddressOf peut souvent être utilisée à la place d'une construction déléguée.
'<typename>' is a delegate type. Delegate construction permits only a single AddressOf expression as an argument list. Often an AddressOf expression can be used instead of a delegate construction.

Une clause **New** qui crée une instance d'une classe déléguée fournit une liste d'arguments non valide au constructeur délégué.

Vous pouvez fournir seulement une expression **AddressOf** lors de la création d'une nouvelle instance déléguée.

Cette erreur peut se produire si vous ne passez au constructeur délégué aucun argument, si vous en passez plusieurs ou un seul qui n'est pas une expression **AddressOf** valide.

ID d'erreur : BC32008

Pour corriger cette erreur

- Utilisez une seule expression **AddressOf** dans la liste d'arguments pour la classe déléguée dans la clause **New**.

Voir aussi

Tâches
Comment : appeler une méthode déléguée
Référence
New (Visual Basic)
AddressOf, opérateur
Concepts
Délégués et opérateur AddressOf

'<nomtype>' est un type de classe et ne peut pas être utilisé en tant qu'expression

'<typename>' is a class type and cannot be used as an expression

Un nom de type apparaît lorsqu'une expression est requise. Une expression doit être constituée d'une combinaison de variables, de constantes, de littéraux, de propriétés et d'appels de procédure **Function**.

ID d'erreur : BC30109

Pour corriger cette erreur

- Supprimez le nom de type et construisez l'expression à l'aide d'éléments valides.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

'<NomType>' porte le même nom qu'un autre type exposé dans un groupe 'My'

Message d'erreur
'<NomType>' porte le même nom qu'un autre type exposé dans un groupe 'My'. Renommez le formulaire ou son espace de noms.
'<typename>' has the same name as another type exposed in a 'My' group. Rename the form or its enclosing namespace.

Une classe ou une structure est déclarée avec le même nom que porte l'une d'entre elles dans l'un des objets **My**.

Une collision de noms n'a pas pu être évitée entre deux classes qui sont accessibles par le biais d'un objet **My**, tel que **My.Forms**.

S'il existe une collision de noms potentielle entre les classes d'un objet **My**, le compilateur modifie le nom de propriété pour le type de *ClassName* en *RootNamespace_Namespace_ClassName*. Par exemple, considérez deux formulaires nommés `Form1`. Si l'un de ces formulaires se trouve dans l'espace de noms racine `WindowsApplication1` et dans l'espace de noms `Namespace1`, vous accéderiez à ce formulaire via `My.Forms.WindowsApplication1_Namespace1_Form1`.

Cette erreur peut se produire si deux classes ont le même nom et se trouvent dans des espaces de noms imbriqués ayant des traits de soulignement dans leurs noms. Lorsque le compilateur construit les nouveaux noms de propriétés pour les classes, il existe toujours une collision de noms.

ID d'erreur : BC36015

Pour corriger cette erreur

1. Renommez le nouveau formulaire.
2. Renommez les espaces de noms.

   Évitez d'attribuer à une classe ou à une structure un nom que porte déjà l'une d'entre elles.

Voir aussi

Référence
My.Forms, objet
Form
MyGroupCollectionAttribute
Concepts
Résolution d'une référence lorsque plusieurs variables ont le même nom

'<nomtype>' ne peut pas masquer une méthode 'MustOverride' déclarée implicitement pour la propriété '<nompropriété>' dans <type> '<nomtype>'

'<typename>' cannot shadow a 'MustOverride' method implicitly declared for property '<propertyname>' in <type> '<typename>'

Le nom de la méthode spécifiée est en conflit avec la méthode **MustOverride** générée implicitement par une propriété dans la classe de base. Par exemple, si vous déclarez une propriété nommée `Prop1`, le compilateur génère les procédures implicites `get_Prop1` et `set_Prop1`.

ID d'erreur : BC31416

Pour corriger cette erreur

1. Attribuez un nom unique à la méthode.
2. Supprimez le modificateur **MustOverride** de la propriété dans la classe de base.

Voir aussi

Référence

MustOverride

Shadows

Concepts

Procédures Property

'<nomtype>' ne peut pas être utilisé en tant qu'attribut, car ce n'est pas une classe

'<typename>' cannot be used as an attribute because it is not a class

Une tentative d'utilisation d'un type autre qu'une classe en tant qu'attribut a été effectuée.

ID d'erreur : BC31503

Pour corriger cette erreur

- Définissez des attributs personnalisés en tant que classes qui dérivent de **System.Attribute**.

Voir aussi

Autres ressources

Attributs en Visual Basic

'<nomtype>' ne peut pas être utilisé en tant qu'attribut, car il est déclaré 'MustInherit'

'<typename>' cannot be used as an attribute because it is declared 'MustInherit'

Les classes d'attributs personnalisés ne peuvent pas être déclarées comme **MustInherit**.

ID d'erreur : BC31506

Pour corriger cette erreur

- Supprimez le modificateur **MustInherit** des attributs personnalisés.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs personnalisés en Visual Basic

'<nomtype>' ne peut pas être utilisé en tant qu'attribut, car il comporte des méthodes 'MustOverride' qui n'ont pas été substituées

'<typename>' cannot be used as an attribute because it has 'MustOverride' methods that have not been overridden

Vous ne pouvez pas utiliser les classes avec des méthodes **MustOverride** en tant qu'attributs.

Vous ne pouvez utiliser les membres**MustOverride** des classes d'attributs qu'à partir de classes dérivées qui se substituent à des membres de ce type.

ID d'erreur : BC31507

Pour corriger cette erreur

1. Supprimez le modificateur **MustOverride** des membres de classes d'attributs.
2. Implémentez les membres **MustOverride** dans une classe dérivée et utilisez cette classe en tant qu'attribut.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs personnalisés en Visual Basic

'<nomtype>' ne peut pas être utilisé en tant qu'attribut, car il n'hérite pas de 'System.Attribute'

'<typename>' cannot be used as an attribute because it does not inherit from 'System.Attribute'

Une tentative d'utilisation d'une classe non dérivée de **System.Attribute** a été effectuée.

ID d'erreur : BC31504

Pour corriger cette erreur

- Définissez des attributs personnalisés en tant que classes dérivant de **System.Attribute** en ajoutant une instruction **Imports** à la première ligne de code de la classe.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs personnalisés en Visual Basic

'<nomtype>' ne peut pas être utilisé en tant qu'attribut, car il ne possède pas un attribut 'System.AttributeUsageAttribute'

'<typename>' cannot be used as an attribute because it does not have a 'System.AttributeUsageAttribute' attribute

Une tentative d'utilisation d'un attribut déclaré sans **System.AttributeUsageAttribute**, nécessaire à la définition de son utilisation, a été effectuée.

ID d'erreur : BC31505

Pour corriger cette erreur

- Les attributs personnalisés doivent être des classes dérivées de **System.Attribute**, auxquelles l'attribut **AttributeUsageAttribute** est appliqué.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs personnalisés en Visual Basic

<nomtype> '<nommembre>' doit être déclaré 'Shadows', car un autre membre du même nom est déclaré 'Shadows'

<typename> '<membername>' must be declared 'Shadows' because another member with this name is declared 'Shadows'

Cette propriété ou procédure possède le même nom qu'une autre propriété ou procédure marquée avec le modificateur **Shadows**.

ID d'erreur : BC30695

Pour corriger cette erreur

- Utilisez le modificateur **Shadows**.

Voir aussi

Référence

Shadows

<type1> '<nomtype>' doit implémenter '<nomméthode>' pour l'interface '<nominterface>'

<type1>'<typename>' must implement '<methodname>' for interface '<interfacename>'

Une classe ou une structure déclare implémenter une interface, mais n'implémente pas une procédure définie par l'interface. Chaque membre de l'interface doit être implémenté.

ID d'erreur : BC30149

Pour corriger cette erreur

1. Déclarez une procédure avec les mêmes nom et signature que ceux définis dans l'interface. Veillez à inclure au moins l'instruction **End Function** ou **End Sub**.
2. Ajoutez une clause **Implements** à la fin de l'instruction **Function** ou **Sub**. Par exemple :

```
Public Sub DoSomething() Implements IBaseInterface.DoSomething
```

Voir aussi

Référence

Implements, instruction

Concepts

Implements, mot clé et instruction

<type1> '<nomtype>' doit implémenter '<nommembre>' pour l'interface '<nominterface>'

Message d'erreur
'<nomtype>' doit implémenter '<nommembre>' pour l'interface '<nominterface>'. La propriété d'implémentation doit avoir des spécificateurs 'ReadOnly'/'WriteOnly' correspondants.
'<typename>' must implement '<membername>' for interface '<interfacename>'. Implementing property must have matching 'ReadOnly'/'WriteOnly' specifiers.

Une classe ou une structure déclare implémenter une interface, mais n'implémente pas une procédure, une propriété ou un événement défini par l'interface. Chaque membre de l'interface doit être implémenté.

ID d'erreur : BC30154

Pour corriger cette erreur

1. Déclarez un membre avec les mêmes nom et signature que ceux définis dans l'interface. Veillez à inclure au moins l'instruction **End Function**, **End Sub** ou **End Property**.
2. Ajoutez une clause **Implements** à la fin de l'instruction **Function**, **Sub**, **Property** ou **Event**. Par exemple :

   ```
   Public Event ItHappened() Implements IBaseInterface.ItHappened
   ```

3. Lors de l'implémentation d'une propriété, vérifiez que **ReadOnly** ou **WriteOnly** est utilisé de la même manière que dans la définition d'interface.
4. Lors de l'implémentation d'une propriété, déclarez les procédures **Get** et **Set** si nécessaire.

Voir aussi

Référence
Implements, instruction
Concepts
Implements, mot clé et instruction

'<type1>' ne peut pas se substituer à <type2>, car il n'est pas déclaré 'Overridable'

'<type1>' cannot override <type2> because it is not declared 'Overridable'

Un membre de classe dérivée se substitue à un membre de classe de base non marqué avec le modificateur **Overridable**.

ID d'erreur : BC31086

Pour corriger cette erreur

1. Ajoutez le modificateur **Overridable** au membre substitué de la classe de base.
2. Utilisez le mot clé **Shadows** pour masquer l'élément dans la classe de base.

Voir aussi

Référence

Overridable
Overrides
Shadows

<type1> '<nomtype1>' est en conflit avec un membre déclaré implicitement pour l'événement '<nomévénement>' dans <type2> '<nomtype2>'

<type1> '<typename1>' conflicts with a member implicitly declared for event '<eventname>' in <type2> '<typename2>'

Le nom d'un membre de type est en conflit avec le nom d'un membre implicitement créé pour un événement. Les événements créent implicitement plusieurs variables implicites. Par exemple, la déclaration `Event X` déclare implicitement les noms `XEventHandler`, `XEvent`, `add_X` et `remove_X`.

ID d'erreur : BC31061

Pour corriger cette erreur

- Renommez le membre déclaré explicitement afin de supprimer le conflit de noms.

Voir aussi

Concepts

Instructions de déclaration en Visual Basic

Autres ressources

Événements en Visual Basic

<type1> '<nomtype>' ne peut pas être déclaré 'Overrides', car il ne se substitue pas à un <type1> dans une base <type2>

<type1> '<typename>' cannot be declared 'Overrides' because it does not override a <type1> in a base <type2>

Une instruction **Sub**, **Function** ou **Property** spécifie **Overrides** lorsqu'il n'existe aucun type du même nom dans une classe de base.

ID d'erreur : BC30284

Pour corriger cette erreur

1. Vérifiez que le nom de type est correctement orthographié.
2. Supprimez le mot clé **Overrides** superflu.

Voir aussi

Concepts

Substitution de propriétés et de méthodes

<type1> '<nomtype>' et <type2> '<nomtype>' (déclarés dans '<nom>') sont en conflit dans l'espace de noms '<nomespacedenoms>'.

<type1> '<typename>' and <type2> '<typename>', declared in '<name>', conflict in <namespace> '<namespacename>'

Deux éléments déclarés, tels qu'une classe et un module, ont le même nom dans le même espace de noms.

ID d'erreur : BC30175

Pour corriger cette erreur

- Déplacez la déclaration de l'un des éléments vers un espace de noms différent.
- Changez le nom de l'un des éléments.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

<type1> '<nomtype>' et <type2> '<nomtype>' sont en conflit dans <espacedenoms> '<nomespacedenoms>'

<type1> '<typename>' and <type2> '<typename>' conflict in <namespace> '<namespacename>'

Deux éléments déclarés, tels qu'une classe et un module, ont le même nom dans le même espace de noms.

ID d'erreur : BC30179

Pour corriger cette erreur

- Déplacez la déclaration de l'un des éléments vers un espace de noms différent.
- Changez le nom de l'un des éléments.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

<type1> '<nompropriété>' est en conflit avec un membre déclaré implicitement pour l'événement '<nomévénement>' dans <type2> '<nomclasse>' de base

<type1> '<propertyname>' conflicts with a member implicitly declared for event '<eventname>' in the base <type2> '<classname>'

Une propriété est déclarée avec le même nom qu'un membre implicite formé à partir d'un événement de la classe de base. Par exemple, si la classe de base définit un événement nommé **Event1**, le compilateur génère les procédures implicites **add_Event1** et **remove_Event1**. Si la propriété dans cette classe a l'un de ces noms, elle doit occulter le membre de la classe de base.

Ce message est un avertissement. **Shadows** est supposé par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40014

Pour corriger cette erreur

1. Pour masquer le membre de la classe de base, ajoutez le mot clé **Shadows** à la déclaration de propriété.
2. Dans le cas contraire, changez le nom de la propriété.

Voir aussi

Référence

Property, instruction
Event, instruction
Shadows

Concepts

Occultation dans Visual Basic

<type1> '<nommembre>' masque un membre surchargeable déclaré dans <type2> '<nomclasse>' de base

Message d'erreur
<type1> '<nommembre>' masque un membre surchargeable déclaré dans <type2> '<nomclasse>' de base. Si vous souhaitez surcharger la méthode de base, vous devez déclarer cette méthode 'Overloads'.
<type1> '<membername>' shadows an overloadable member declared in the base <type2> '<classname>. If you want to overload the base method, this method must be declared 'Overloads'.

Une classe dérivée définit une procédure **Function** ou **Sub**, ou **Property** avec le même nom qu'une procédure ou une propriété définie dans la classe de base. Étant donné que les procédures et les propriétés sont des membres surchargeables, la classe dérivée peut surcharger ou masquer le membre de la classe de base. Toutefois, le code de la classe dérivée ne spécifie ni Overloads ni Shadows dans la déclaration. Dans l'absence de l'un ou l'autre des mots clés, le compilateur suppose **Shadows**.

En programmation, il est conseillé de spécifier **Overloads** ou **Shadows**. Cela rend votre code plus lisible et plus compréhensible.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40003

Pour corriger cette erreur

- Si vous souhaitez surcharger la méthode ou la propriété de la classe de base, incluez le mot clé **Overloads** dans la déclaration.
- Si vous souhaitez masquer la méthode ou la propriété de la classe de base, incluez le mot clé **Shadows** à la place de **Overloads**.
- Si vous ne souhaitez ni surcharger ni masquer le membre de la classe de base, modifiez le nom du membre de la classe dérivée.

Voir aussi

Référence
Overloads
Shadows
Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)
Property, instruction

Concepts
Surcharge de procédure
Occultation dans Visual Basic

<type1> '<nommembre>' est en conflit avec <type2> '<nommembre>' dans la classe <type3> '<nomclasse>' de base et doit donc être déclaré 'Shadows'

<type1> '<membername>' conflicts with <type2> '<membername>' on the base class <type3> '<classname>' and should be declared 'Shadows'

Un élément de programmation est déclaré avec le même nom qu'un élément défini dans la classe de base. Dans ce cas, l'élément de cette classe doit occulter l'élément de la classe de base.

Ce message est un avertissement. **Shadows** est supposé par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40004

Pour corriger cette erreur

- Ajoutez le mot clé **Shadows** à la déclaration ou changez le nom de l'élément déclaré.

Voir aussi

Référence

Shadows

Concepts

Occultation dans Visual Basic

Les paramètres <type> ne peuvent pas être déclarés 'ParamArray'

<type> parameters cannot be declared 'ParamArray'

Une définition d'un délégué, d'un événement ou d'un opérateur déclare un paramètre ParamArray.

Les paramètres **ParamArray** ne sont autorisés que sur les paramètres **Declare**, **Function**, **Property** et **Sub**.

ID d'erreur : BC33009

Pour corriger cette erreur

- Supprimez le mot clé **ParamArray** de la liste de paramètres.
- Si vous définissez un opérateur, vous pouvez être en mesure d'obtenir les fonctionnalités **ParamArray** avec une série de surcharges.
- Si vous définissez un délégué ou un événement, vous devez revoir toute la logique de cette partie de votre application. Vous ne pouvez pas utiliser les paramètres Optional (Visual Basic) ou **ParamArray**, ou encore des versions surchargées sur les paramètres de délégués ou d'événements.

Voir aussi

Référence

Overloads

Operator, instruction

Concepts

Procédures d'opérateur

Les paramètres <type> ne peuvent pas être déclarés 'Optional'

<type> parameters cannot be declared 'Optional'

Une définition d'un délégué, d'un événement ou d'un opérateur déclare un paramètre Optional (Visual Basic).

Les paramètres **Optional** ne sont autorisés que sur les paramètres **Declare**, **Function**, **Property** et **Sub**.

ID d'erreur : BC33010

Pour corriger cette erreur

- Supprimez le mot clé **Optional** de la liste de paramètres.
- Si vous définissez un opérateur, vous pouvez être en mesure d'obtenir les fonctionnalités **Optional** avec une série de surcharges.
- Si vous définissez un délégué ou un événement, vous devez revoir toute la logique de cette partie de votre application. Vous ne pouvez pas utiliser les paramètres **Optional** ou ParamArray, ou encore des versions surchargées sur les paramètres de délégués ou d'événements.

Voir aussi

Référence

Overloads

Operator, instruction

Concepts

Procédures d'opérateur

'<type>' ne peut pas être hérité plus d'une fois

'<type>' cannot be inherited more than once

Vous avez dupliqué une instruction **Inherits**.

ID d'erreur : BC30584

Pour corriger cette erreur

- Supprimez l'instruction **Inherits** redondante.

Voir aussi

Référence

Inherits, instruction

<type> '<nomtype>' masque une méthode substituable dans une classe de base

Message d'erreur
<type> '<nomtype>' masque une méthode substituable dans une classe de base. Pour substituer la méthode de base, vous devez déclarer cette méthode 'Overrides'.
<type> '<typename>' shadows an overridable method in the base class. If you want to override the base method, this method must be declared 'Overrides'.

Un élément de programmation est déclaré avec le même nom qu'une procédure ou une propriété substituable définie dans la classe de base. Dans ce cas, l'élément de cette classe doit occulter l'élément de la classe de base.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40005

Pour corriger cette erreur

- Si vous souhaitez substituer la procédure de base, ajoutez le mot clé **Overrides** à la déclaration.
- Si vous souhaitez occulter la procédure de base, ajoutez le mot clé **Shadows** à la déclaration.
- Si vous ne souhaitez ni substituer ni occulter la procédure de base, changez le nom de l'élément déclaré.

Voir aussi

Référence
Overrides
Shadows
Concepts
Substitution de propriétés et de méthodes
Occultation dans Visual Basic

<type> '<nomméthode>' est en conflit avec d'autres membres portant le même nom dans toute la hiérarchie d'héritage et doit donc être déclaré 'Shadows'

<type> '<methodname>' conflicts with other members of the same name across the inheritance hierarchy and so should be declared 'Shadows'

Une interface qui hérite de plusieurs interfaces définit une procédure avec le même nom qu'une procédure déjà définie dans plusieurs interfaces de base. La procédure dans cette interface doit occulter l'une des procédures de l'interface de base.

Ce message est un avertissement. **Shadows** est supposé par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42000

Pour corriger cette erreur

- Si vous souhaitez masquer l'une des procédures d'interface de base, ajoutez le mot clé **Shadows** à la nouvelle déclaration de procédure.
- Dans le cas contraire, changez le nom de la nouvelle procédure.

Voir aussi

Référence

Shadows

Concepts

Occultation dans Visual Basic

L'instruction '<nominstruction>' requiert un tableau

'<statementname>' statement requires an array

Les instructions **Erase** et **ReDim** requièrent des types de données tableau. Le type fourni n'est pas un tableau.

ID d'erreur : BC30049

Pour corriger cette erreur

- Remplacez le type de données par un type tableau.

Voir aussi

Référence

Erase, instruction (Visual Basic)

ReDim, instruction (Visual Basic)

Autres ressources

Tableaux dans Visual Basic

Impossible de combiner '<spécificateur1>' et '<spécificateur2>'

'<specifier1>' and '<specifier2>' cannot be combined

Vous utilisez ensemble deux spécificateurs incompatibles, tels que **Shadows** et **Overrides**.

ID d'erreur : BC31408

Pour corriger cette erreur

- Supprimez l'un ou l'autre des spécificateurs.

Voir aussi

Référence

Shadows
Overrides

<spécificateur1> <type> ne peut pas hériter de <spécificateur2> <type> car il étend l'accès de la base <type>

<specifier1> <type> cannot inherit from a <specifier2> <type> because it expands the access of the base <type>

Vous avez essayé qu'une classe publique hérite d'une classe de base marquée comme **Private** ou **Friend**.

ID d'erreur : BC30509

Pour corriger cette erreur

- Déclarez la classe de base **Public** ou la classe qui hérite de **Private** ou **Friend**.

Voir aussi

Autres ressources

Héritage en Visual Basic

'<spécificateur>' n'est pas valide dans une déclaration de propriété d'interface

'<specifier>' is not valid on an interface property declaration

Une instruction **Property** à l'intérieur d'une interface contient un mot clé non valide, tel que **Implements**. Une interface peut seulement définir des membres, elle ne peut pas les implémenter.

ID d'erreur : BC30273

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction de déclaration.
- Déplacez l'implémentation des membres d'interface vers une classe qui implémente l'interface.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Implements, instruction

'<spécificateur>' n'est pas valide dans une déclaration de méthode d'interface

'<specifier>' is not valid on an interface method declaration

Une instruction **Function** ou **Sub** à l'intérieur d'une interface contient un mot clé non valide, tel que **Implements**. Une interface peut seulement définir des membres, elle ne peut pas les implémenter.

ID d'erreur : BC30270

Pour corriger cette erreur

1. Supprimez le mot clé non valide de l'instruction de déclaration.
2. Déplacez l'implémentation des membres d'interface vers une classe qui implémente l'interface.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Implements, instruction

'<spécificateur>' n'est pas valide dans une déclaration event d'interface

'<specifier>' is not valid on an interface event declaration

Une instruction **Event** à l'intérieur d'une interface contient un mot clé non autorisé, tel que **Implements**. Une interface peut seulement définir des membres, elle ne peut pas les implémenter.

ID d'erreur : BC30275

Pour corriger cette erreur

1. Supprimez le mot clé de l'instruction de déclaration.
2. Déplacez l'implémentation des membres d'interface vers une classe qui implémente l'interface.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Implements, instruction

'<spécificateur>' n'est pas valide dans une déclaration Enum

'<specifier>' is not valid on an Enum declaration

Vous avez utilisé un spécificateur non valide dans une déclaration **Enum**.

ID d'erreur : BC30396

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Concepts

Vue d'ensemble des énumérations

'<spécificateur>' n'est pas valide dans une déclaration WithEvents

'<specifier>' is not valid on a WithEvents declaration

Une instruction **Dim** contient un mot clé, tel que **ReadOnly**, qui est non valide en combinaison avec le mot clé **WithEvents**.

ID d'erreur : BC30234

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Dim**.

Voir aussi

Référence

Dim, instruction (Visual Basic)
WithEvents

'<spécificateur>' n'est pas valide dans une déclaration Structure

'<specifier>' is not valid on a Structure declaration

Vous avez utilisé un spécificateur (par exemple, **Dim** ou **Override**) qui n'est pas valide dans une déclaration **Structure**.

ID d'erreur : BC30395

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Concepts

Structures et classes

'<spécificateur>' n'est pas valide dans une déclaration de variable membre

'<specifier>' is not valid on a member variable declaration

Une instruction **Dim** contient un mot clé non valide. Une instruction **Dim** ne peut inclure que les mots clés **Friend**, **Private**, **Protected**, **Public**, **ReadOnly**, **Shadows**, **Shared** ou **Static**.

Ce message peut également s'afficher si vous déclarez une variable **Static** en dehors d'une procédure. Vous pouvez utiliser **Static** seulement au niveau de la procédure.

Notez que si vous incluez un mot clé valide dans une instruction **Dim**, vous pouvez éventuellement omettre le mot clé **Dim**.

ID d'erreur : BC30235

Pour corriger cette erreur

1. Supprimez le mot clé non valide de l'instruction **Dim**.
2. Si vous avez déclaré une variable **Static** à l'extérieur d'une procédure, déplacez la déclaration à l'intérieur de la procédure ou supprimez le mot clé **Static**.

Voir aussi

Référence

Dim, instruction (Visual Basic)

'<spécificateur>' n'est pas valide dans une déclaration de constante

'<specifier>' is not valid on a constant declaration

Une instruction **Const** contient un mot clé non valide. Une instruction **Const** ne peut inclure que les mots clés **Friend**, **Private**, **Protected**, **Public** ou **Shadows**.

ID d'erreur : BC30233

Pour corriger cette erreur

1. Vérifiez que le mot clé est correctement orthographié.
2. Supprimez le mot clé non valide de l'instruction **Const**.

Voir aussi

Référence

Const, instruction (Visual Basic)

'<rsetstmt>' n'est pas déclaré

Message d'erreur
'<rsetstmt>' n'est pas déclaré. Les instructions RSet ne sont plus prises en charge ; utilisez Microsoft.VisualBasic.RSet à la place.
'<rsetstmt>' is not declared. RSet statements are no longer supported; use Microsoft.VisualBasic.RSet instead.

Plusieurs fonctions qui étaient intrinsèques à Visual Basic dans les versions antérieures ont été déplacées dans l'espace de noms Microsoft.VisualBasic. En conséquence, leurs fonctionnalités sont plus accessibles aux langages de programmation.

ID d'erreur : BC30821

Pour corriger cette erreur

- Utilisez à la place la fonction **RSet** dans l'espace de noms **Microsoft.VisualBasic**.

Voir aussi

Référence
RSet, fonction
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

'<méthodequalifiée>' n'est pas accessible dans ce contexte, car elle est '<modificateur>'

'<qualifiedmethod>' is not accessible in this context because it is '<modifier>'

Vous avez tenté d'accéder à une méthode qui n'est pas accessible dans ce contexte car un modificateur lui a été appliqué.

ID d'erreur : BC30390

Pour corriger cette erreur

- Supprimez le modificateur.

Voir aussi

Référence

Private (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)

Concepts

Considérations sur la sélection de niveaux d'accès pour les méthodes

'<nompropriété1>' ne peut pas se substituer à '<nompropriété2>', car 'ReadOnly' ou 'WriteOnly' les différencie

'<propertyname1>' cannot override '<propertyname2>' because they differ by 'ReadOnly' or 'WriteOnly'

Vous avez tenté de substituer une propriété par une autre qui diffère uniquement de la première par **ReadOnly** ou **WriteOnly**.

ID d'erreur : BC30362

Pour corriger cette erreur

- Vérifiez que les propriétés ne se distinguent pas uniquement par **ReadOnly** et **WriteOnly**.

Voir aussi

Référence

Overrides

Concepts

Substitution de propriétés et de méthodes

'<nompropriété>' définit implicitement '<nommembre>', qui est en conflit avec un membre portant le même nom dans <type> '<nomtype>'

'<propertyname>' implicitly defines '<membername>', which conflicts with a member of the same name in <type> '<typename>'

Le nom d'un membre de type est en conflit avec le nom d'un membre implicitement créé pour un événement. Les propriétés créent implicitement plusieurs variables implicites qui peuvent être en conflit avec des membres définis.

ID d'erreur : BC31060

Pour corriger cette erreur

- Renommez le membre déclaré explicitement afin de supprimer le conflit de noms.

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Concepts

Procédures Property

Propriétés et procédures Property

'<nompropriété>' ne peut pas être exposé à COM en tant que propriété 'Let'

Message d'erreur
'<nompropriété>' ne peut pas être exposé à COM en tant que propriété 'Let'. Vous ne serez pas en mesure d'assigner des valeurs qui ne sont pas des objets (telles que des nombres ou des chaînes) à cette propriété à partir de Visual Basic 6.0 à l'aide d'une instruction 'Let'.
'<propertyname>' cannot be exposed to COM as a property 'Let'. You will not be able to assign non-object values (such as numbers or strings) to this property from Visual Basic 6.0 using a 'Let' statement.

Une classe qui utilise un bloc d'attributs **COMClassAttribute** déclare une propriété **Public** avec un type de données **Object**. Un programme Visual Basic 6.0 peut accéder à cette propriété en tant que **Variant**, mais ne peut lui assigner qu'une référence d'objet avec l'instruction **Set**. Il ne peut pas assigner un type valeur avec l'instruction **Let**.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42102

Pour traiter cet avertissement

- Nous vous recommandons d'avertir les utilisateurs Visual Basic 6.0 potentiels de cette classe qu'ils ne peuvent pas utiliser cette propriété avec l'instruction **Let**.

Voir aussi

Référence
Property, instruction
Public (Visual Basic)
Object, type de données
ComClassAttribute, classe

Concepts
Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0
Attributs utilisés dans Visual Basic
Application des attributs

'<propriété1>' et '<propriété2>' ne peuvent pas se surcharger mutuellement, car seule l'une d'entre elles est déclarée 'Default'

'<property1>' and '<property2>' cannot overload each other because only one is declared 'Default'

Si une propriété comprend le modificateur **Default**, toutes les propriétés surchargées sur ce nom doivent également spécifier **Default**.

ID d'erreur : BC30361

Pour corriger cette erreur

- Vérifiez que toutes les propriétés surchargées sont déclarées **Default**.

Voir aussi

Référence

Default (Visual Basic)

Concepts

Considérations sur les surcharges de procédures

Les paramètres <Propriété> ne peuvent pas être déclarés 'ByRef'

<Property> parameters cannot be declared 'ByRef'

Les paramètres de propriété doivent être déclarés à l'aide du modificateur **ByVal**.

ID d'erreur : BC30651

Pour corriger cette erreur

- Définissez des paramètres de propriété en tant que **ByVal**.

Voir aussi

Référence

ByVal
ByRef

<SignatureProcédure1> n'est pas conforme au CLS, car il surcharge <SignatureProcédure2> dont il diffère uniquement par un tableau de types de paramètres tableau ou par la dimension des types de paramètres tableau.

<proceduresignature1> is not CLS-compliant because it overloads <proceduresignature2> which differs from it only by array of array parameter types or by the rank of the array parameter types

Une procédure ou une propriété est marquée comme `<CLSCompliant(True)>` lorsqu'elle substitue une autre procédure ou propriété et que la seule différence entre leurs listes de paramètres est le niveau d'imbrication d'un tableau en escalier ou le rang d'un tableau.

Dans les déclarations suivantes, les deuxième et troisième déclarations génèrent cette erreur.

```
Overloads Sub processArray(ByVal arrayParam() As Integer)

Overloads Sub processArray(ByVal arrayParam()() As Integer)

Overloads Sub processArray(ByVal arrayParam(,) As Integer)
```

La deuxième déclaration remplace le paramètre unidimensionnel d'origine `arrayParam` par un tableau de tableaux. La troisième déclaration remplace `arrayParam` par un tableau à deux dimensions (rang 2). Bien que Visual Basic autorise les surcharges à différer uniquement par l'une de ces modifications, une telle surcharge n'est pas conforme au Spécification CLS (Common Language Specification) (CLS).

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40035

Pour corriger cette erreur

- Si la conformité CLS est requise, définissez vos surcharges pour qu'elles diffèrent les unes des autres autrement que par les seules modifications mentionnées dans cette page d'aide.
- Si les surcharges doivent différer uniquement par les modifications mentionnées dans cette page d'aide, supprimez **CLSCompliantAttribute** de leurs définitions ou marquez-les comme `<CLSCompliant(False)>`.

Voir aussi

Référence

Overloads

Concepts

Écriture d'un code conforme CLS

Surcharge de procédure

<nomprocédure1> ne peut pas se substituer à <nomprocédure2>, car ils diffèrent sur les paramètres déclarés 'ParamArray'

<procedurename1> cannot override <procedurename2> because they differ by parameters declared 'ParamArray'

Une procédure dans une classe dérivée substitue une procédure de même nom dans une classe de base, mais les listes de paramètres sont différentes.

Pour substituer une procédure dans une classe héritée, la procédure de substitution doit correspondre à sa liste de paramètres, son niveau d'accès et son type de retour (éventuel). En particulier, elle doit correspondre à tout Optional (Visual Basic) ou déclaration ParamArray.

ID d'erreur : BC30906

Pour corriger cette erreur

- Si vous souhaitez substituer la procédure, faites que la liste de paramètres soit exactement la même que la liste de paramètres dans la procédure de classe de base. Si le dernier paramètre est déclaré avec **ParamArray** dans la procédure de classe de base, déclarez-le avec **ParamArray** dans la procédure de substitution.
- Si vous souhaitez une liste de paramètres différente de la version de classe de base, vous ne pouvez pas la substituer. Vous pouvez la surcharger à la place. Pour plus d'informations, consultez Surcharge de procédure.

Voir aussi

Référence

Overrides

Concepts

Substitution de propriétés et de méthodes

'<NomProcédure1>' ne peut pas se substituer à '<NomProcédure2>', car il n'est pas accessible dans ce contexte

'<procedurename1>' cannot override '<procedurename2>' because it is not accessible in this context

Une procédure ou une propriété substitue une procédure ou une propriété avec un niveau d'accès qui empêche l'accès par la procédure ou la propriété de substitution.

Par exemple, si une procédure est déclarée comme **Friend** dans un assembly, elle n'est plus accessible à l'extérieur de cet assembly. Si une procédure d'un autre assembly du même projet essaie de substituer la procédure **Friend**, elle ne peut pas y accéder pour la substituer.

ID d'erreur : BC31417

Pour corriger cette erreur

- Déplacez la procédure ou la propriété de substitution dans le même assembly que la procédure ou la propriété qu'elle doit substituer.

  - ou -

- Supprimez le mot clé **Overrides**.

Voir aussi

Référence

Overrides

Concepts

Niveaux d'accès dans Visual Basic

Substitution de propriétés et de méthodes

La méthode '<NomProcédure>' pour l'événement '<NomÉvénement>' ne peut pas être marquée comme conforme CLS, car son type conteneur '<NomType>' n'est pas conforme CLS

'<procedurename>' method for event '<eventname>' cannot be marked CLS compliant because its containing type '<typename>' is not CLS compliant

Un événement personnalisé déclare une procédure **AddHandler** ou **RemoveHandler** et la marque comme `<CLSCompliant(True)>`, mais l'événement est défini dans un type qui est marqué comme `<CLSCompliant(False)>` ou qui n'est pas marqué.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40053

Pour corriger cette erreur

- Si la conformité CLS est requise, définissez l'événement dans un type conforme CLS.
- Si l'événement doit rester dans son type conteneur, supprimez **CLSCompliantAttribute** de sa définition ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Tâches

Comment : déclarer des événements qui évitent les blocages
Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Concepts

AddHandler et RemoveHandler
Écriture d'un code conforme CLS

'<nomprocédure>' n'a aucun paramètre et son type de retour ne peut pas être indexé

'<procedurename>' has no parameters and its return type cannot be indexed

Un appel à une procédure **Function** ou **Sub** fournit un ou plusieurs arguments, mais la procédure ne prend pas d'argument, et son type de retour (s'il s'agit de **Function**) n'est pas un type tableau.

ID d'erreur : BC32016

Pour corriger cette erreur

- Supprimez l'argument ou les arguments de l'appel de procédure.

Voir aussi

Référence

Function, instruction (Visual Basic)

Sub, instruction (Visual Basic)

Concepts

Sub, procédures

Function, procédures

'<nomprocédure>' a un type de retour ou des types de paramètres non pris en charge

'<procedurename>' has a return type that is not supported or parameter types that are not supported

Une procédure est déclarée avec des paramètres ou un type de retour ne pouvant pas être représentés par Visual Studio.

ID d'erreur : BC30657

Pour corriger cette erreur

- Définissez les types de retour ou les paramètres en tant que type pris en charge.
- Développez un nouveau type qui décrit les données utilisées par cette procédure.

Voir aussi

Référence

Liste des types de données (Visual Basic)

'<NomProcédure>' ne peut pas se substituer à '<NomProcédureDeBase>', car leurs contraintes de paramètre de type diffèrent

'<procedurename>' cannot override '<baseprocedurename>' because they differ by type parameter constraints

Une procédure générique essaie de substituer une procédure de classe de base générique, mais elles possèdent des listes de contraintes différentes sur leurs paramètres de type.

Pour substituer une procédure de classe de base, la procédure de substitution doit correspondre non seulement à la signature complète de la procédure de classe de base, mais également au niveau d'accès de la procédure et au mécanisme de passage de chaque paramètre.

Pour substituer une procédure de classe de base générique, la procédure de substitution doit également correspondre au nombre de paramètres de type et à la liste de contraintes de chacun d'eux.

Pour plus d'informations sur les exigences de substitution, consultez Overrides.

ID d'erreur : BC32077

Pour corriger cette erreur

- Si vous comptez substituer la procédure de classe de base, modifiez les contraintes de paramètre de type pour qu'elles correspondent exactement à celles de la procédure de classe de base.
- Si les contraintes de paramètre de type doivent rester telles quelles, vous ne pouvez pas substituer la procédure de classe de base. Supprimez le mot clé **Overrides** de la déclaration.

Voir aussi

Concepts

Types génériques en Visual Basic

'<procédure1>' et '<procédure2>' ne peuvent pas se surcharger mutuellement, car seuls les paramètres déclarés 'ByRef' ou 'ByVal' les différencient

Message d'erreur
'<procédure1>' et '<procédure2>' ne peuvent pas se surcharger mutuellement, car seuls les paramètres déclarés ByRef ou ByVal les différencient. Shadows est pris en compte par défaut.
'<procedure1>' and '<procedure2>' cannot overload each other because they differ only by parameters declared ByRef or ByVal. Shadows assumed.

Deux déclarations de procédure spécifient le même nom et la même liste d'arguments, et la seule différence réside dans les caractéristiques de **ByRef** ou **ByVal** pour un ou plusieurs arguments. Les versions surchargées d'une procédure doivent différer l'une de l'autre par le nombre, l'ordre ou les types de données des arguments.

Ce message est un avertissement. **Shadows** est supposé par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez
Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42003

Pour corriger cette erreur

- Si vous souhaitez créer un ensemble de versions surchargées d'une procédure, faites en sorte que le nombre, l'ordre ou les types de données d'arguments soient différents dans chaque version. Par ailleurs, ajoutez le mot clé **Overloads** à chaque déclaration.
- Si vous ne souhaitez pas surcharger une procédure, changez le nom de la procédure dans l'une des déclarations.

Voir aussi

Concepts
Surcharge de procédure

'<NomParamètre>' est déjà déclaré en tant que paramètre de type de cette méthode

'<parametername>' is already declared as a type parameter of this method

Une procédure générique définit un paramètre normal ou une variable locale avec le même nom qu'un paramètre de type.

Chaque paramètre d'une procédure, y compris chaque paramètre de type d'une procédure générique, doit posséder un nom différent de tous les autres paramètres. Étant donné que les paramètres de procédure sont utilisés comme des variables locales, toute variable locale déclarée dans la procédure doit également posséder un nom différent de tous les paramètres et des paramètres de type.

ID d'erreur : BC32089

Pour corriger cette erreur

- Modifiez le nom du paramètre normal ou de la variable locale.

Voir aussi

Référence

Liste de paramètres

Concepts

Procédures génériques dans Visual Basic

'<constantenull>' n'est pas déclaré

Message d'erreur
'<constantenull>' n'est pas déclaré. La constante Null n'est plus prise en charge ; utilisez System.DBNull à la place.
'<nullconstant>' is not declared. Null constant is no longer supported; use System.DBNull instead.

Une instruction utilise le mot clé **Null** qui n'est plus pris en charge dans Visual Basic 2005.

ID d'erreur : BC30822

Pour corriger cette erreur

1. Utilisez DBNull au lieu de **Null**. C'est ce que montre l'exemple suivant.

```
Sub TestDBNull()
    Dim t As DataTable
    ' Assume the DataGrid is bound to a DataTable.
    t = CType(DataGrid1.DataSource, DataTable)
    Dim r As DataRow
    r = t.Rows(datagrid1.CurrentCell.RowNumber)
    r.BeginEdit
    r(1) = System.DBNull.Value ' Assign DBNull to the record.
    r.EndEdit
    r.AcceptChanges
    If r.IsNull(1) Then
        MsgBox("")
    End If
End Sub
```

2. Utilisez le mot clé Nothing (Visual Basic) pour les devoirs et les comparaisons lorsque vous utilisez des variables Object. C'est ce que montre l'exemple suivant.

```
Sub TestNothing()
    Dim cls As Object
    ' cls is Nothing if it has not been assigned using the New keyword.
    If (cls Is Nothing) Then
        MsgBox("cls is Nothing")
    End If
    cls = Nothing ' Assign Nothing to the class variable cls.
End Sub
```

Voir aussi

Référence
Nothing (Visual Basic)
DBNull
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

'<nomespacedenoms>' est ambigu

'<namespacename>' is ambiguous

Vous avez fourni un nom ambigu qui entre par conséquent en conflit avec un autre nom. Le compilateur Visual Basic ne possède aucune règle de résolution de conflit ; vous devez lever vous-même toute ambiguïté sur les noms.

ID d'erreur : BC30554

Pour corriger cette erreur

- Qualifiez le nom de manière complète.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

'<espacedenoms>' n'est pas un nom valide et ne peut pas être utilisé comme nom d'espace de noms racine

'<namespace>' is not a valid name and cannot be used as the root namespace name

Un nom d'espace de noms qualifiant n'est pas un nom d'élément valide. Un nom doit contenir uniquement des caractères alphabétiques, des nombres décimaux et des traits de soulignement, et il doit commencer par un caractère alphabétique ou un trait de soulignement (_).

ID d'erreur : BC30113

Pour corriger cette erreur

- Si l'espace de noms fait partie d'un ensemble d'espaces de noms imbriqués, vérifiez si les noms externes contiennent des caractères non valides.

Voir aussi

Concepts

Espaces de noms dans Visual Basic
Noms d'éléments déclarés

'<nom1>' est ambigu, importé des espaces de noms ou des types '<nom2>'

'<name1>' is ambiguous, imported from the namespaces or types '<name2>'

Vous avez fourni un nom ambigu qui entre par conséquent en conflit avec un autre nom. Le compilateur Visual Basic ne possède aucune règle de résolution de conflit ; vous devez lever vous-même toute ambiguïté sur les noms.

ID d'erreur : BC30561

Pour corriger cette erreur

1. Levez toute ambiguïté sur le nom en supprimant les importations d'espaces de noms.
2. Qualifiez le nom de manière complète.

Voir aussi

Référence

Imports, instruction

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

Le nom '<nom1>' pour les Imports '<nom2>' ne fait pas référence à une déclaration Namespace, Class, Structure, Enum ou Module

'<name1>' for the Imports '<name2>' does not refer to a Namespace, Class, Structure, Enum or Module

Vous avez essayé d'utiliser l'instruction **Imports** sur quelque chose autre que **Namespace**, **Class**, **Structure**, **Enum** ou **Module**. L'instruction **Imports** importe des noms d'espaces de noms à partir de projets et d'assemblys référencés, ou des noms d'espaces de noms définis dans le même projet que le module dans lequel apparaît l'instruction.

ID d'erreur : BC30467

Pour corriger cette erreur

- Vérifiez l'entité que vous essayez d'importer et assurez-vous qu'elle est valide pour une utilisation avec une instruction **Imports**.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Référence

Imports, instruction

Concepts

Références et l'instruction Imports

'<nom1>' est en conflit avec un <type> portant le même nom défini dans '<nom2>'

'<name1>' conflicts with a <type> by the same name defined in '<name2>'

Le nom rencontré est en conflit avec un type portant le même nom.

ID d'erreur : BC31073

Pour corriger cette erreur

1. Déclarez un nom unique.
2. Mettez le nom du membre en conflit entre crochets.

Voir aussi

Autres ressources

Structure de programme et conventions de codage

'<nom>', qui est nécessaire pour la compilation de cette construction, est introuvable

'<name>', necessary for compiling this construct, cannot be found

Un élément nécessaire à la compilation reste introuvable.

ID d'erreur : BC30930

Pour corriger cette erreur

1. Fournissez l'élément d'assistance nécessaire.
2. Assurez-vous que le code est du code managé valide.

Voir aussi

Concepts

Processus d'exécution managée

'<nom>' n'est pas un membre de '<nomclasse>'

'<name>' is not a member of '<classname>'

Le membre fourni n'est pas un membre de la classe.

ID d'erreur : BC30456

Pour corriger cette erreur

1. Vérifiez que le nom du membre est correct.
2. Utilisez un membre réel de la classe.

Voir aussi

Concepts

Méthodes de classe

'<nom>' est déclaré dans le projet '<nomprojet1>', qui n'est pas référencé par le projet '<nomprojet2>'

'<name>' is declared in project '<projectname1>', which is not referenced by project '<projectname2>'

Un élément de programmation est défini dans un projet auquel le projet en cours ne fait pas référence.

ID d'erreur : BC32004

Pour corriger cette erreur

- Dans le projet en cours, ajoutez une référence au projet contenant l'élément souhaité.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Référence

/reference (Visual Basic)

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

'<nom>' est ambigu dans l'espace de noms '<nomespacedenoms>'

'<name>' is ambiguous in the namespace '<namespacename>'

Vous avez fourni un nom ambigu qui entre par conséquent en conflit avec un autre nom. Le compilateur Visual Basic ne possède aucune règle de résolution de conflit ; vous devez lever vous-même toute ambiguïté sur les noms.

ID d'erreur : BC30560

Pour corriger cette erreur

- Qualifiez le nom de manière complète.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

'<nom>' est ambigu dans les objets application '<liste>'

'<name>' is ambiguous in the application objects '<list>'

Vous avez fourni un nom ambigu qui entre par conséquent en conflit avec un autre nom. Le compilateur Visual Basic ne possède aucune règle de résolution de conflit ; vous devez lever vous-même toute ambiguïté sur les noms. La liste fournie dans le message d'erreur détaille les objets application dans lesquels le nom est ambigu.

ID d'erreur : BC30563

Pour corriger cette erreur

- Qualifiez le nom de manière complète.

Voir aussi

Référence

Namespace, instruction

'<nom>' est ambigu entre les déclarations dans les modules '<nommodule1>' et '<nommodule2>'

'<name>' is ambiguous between declarations in Modules '<modulename1>' and '<modulename2>'

Vous avez fourni un nom ambigu qui entre par conséquent en conflit avec un autre nom. Le compilateur Visual Basic ne possède aucune règle de résolution de conflit ; vous devez lever vous-même toute ambiguïté sur les noms.

ID d'erreur : BC30562

Pour corriger cette erreur

- Qualifiez le nom de manière complète.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

'<nom>' est déjà déclaré en tant que paramètre de cette méthode

'<name>' is already declared as a parameter of this method

Le nom de paramètre spécifié a déjà été déclaré pour cette méthode.

ID d'erreur : BC30734

Pour corriger cette erreur

- Renommez le paramètre de manière unique.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

'<nom>' est déjà déclaré en tant que '<déclaration>' dans cet <espacededéclaration>

'<name>' is already declared as '<declaration>' in this <declarationspace>

Deux éléments de programmation sont déclarés avec le même nom dans le même espace de déclaration (classe, module, interface ou structure).

ID d'erreur : BC30260

Pour corriger cette erreur

- Déplacez une des déclarations vers un espace de déclaration différent ou changez le nom de l'un des éléments déclarés.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

'<nom>' ne peut pas exposer le type '<type1>' utilisé dans le type délégué sous-jacent '<typedélégué>' de l'événement qu'il implémente en dehors du projet via <spécificateur> '<type2>'

'<name>' cannot expose type '<type1>' used in the underlying delegate type '<delegatetype>' of the event it is implementing outside the project through <specifier> '<type2>'

Le code expose un type privé à l'extérieur de la classe publique.

ID d'erreur : BC30924

Pour corriger cette erreur

- Déclarez le type **Public** ou utilisez un autre type.

Voir aussi

Autres ressources

Programmation orientée objet dans Visual Basic

'<nom>' ne peut pas exposer un type <type1> dans <spécificateur> <type2> '<nomtype>' à travers <nom>

'<name>' cannot expose type <type1> in <specifier> <type2> '<typename>' through <name>

Vous avez tenté d'exposer un type privé en dehors de la classe publique **MyClass**.

ID d'erreur : BC30508

Pour corriger cette erreur

- Déclarez le type **Public**.

Voir aussi

Référence

MyClass

'<nom>' ne peut pas exposer le type '<type>' utilisé dans le type délégué '<typedélégué>' sous-jacent de l'événement qu'il implémente à <spécificateur1> '<type1>' via <spécificateur2> '<type2>'

'<name>' cannot expose type '<type>' used in the underlying delegate type '<delegatetype>' of the event it is implementing to <specifier1> '<type1>' through <specifier2> '<type2>'

Le code expose un type privé à l'extérieur de la classe publique.

ID d'erreur : BC30923

Pour corriger cette erreur

- Déclarez le type **Public** ou utilisez un autre type.

Voir aussi

Autres ressources

Programmation orientée objet dans Visual Basic

'<nom>' ne peut pas exposer le type délégué '<typedélégué>' sous-jacent de l'événement qu'il implémente à <spécificateur1> '<type1>' via <spécificateur2> '<type2>'

'<name>' cannot expose the underlying delegate type '<delegatetype>' of the event it is implementing to <specifier1> '<type1>' through <specifier2> '<type2>'

Le code expose un type privé à l'extérieur de la classe publique.

ID d'erreur : BC30914

Pour corriger cette erreur

- Déclarez le type **Public** ou utilisez un autre type.

Voir aussi

Autres ressources

Programmation orientée objet dans Visual Basic

'<nom>' ne peut pas exposer le type délégué '<typedélégué>' sous-jacent de l'événement qu'il implémente en dehors du projet via <spécificateur> '<type>'

'<name>' cannot expose the underlying delegate type '<delegatetype>' of the event it is implementing outside the project through <specifier> '<type>'

Le code expose un type privé à l'extérieur de la classe publique.

ID d'erreur : BC30915

Pour corriger cette erreur

- Déclarez le type **Public**.

  - ou -

- Utilisez un autre type.

Voir aussi

Autres ressources

Programmation orientée objet dans Visual Basic

'<nom>' ne peut pas être nommé en tant que paramètre dans un spécificateur d'attribut, car ce n'est ni un champ ni une propriété

'<name>' cannot be named as a parameter in an attribute specifier because it is not a field or property

Un bloc d'attributs définit une valeur pour un membre qui n'est pas une variable de l'attribut. Vous pouvez assigner des valeurs uniquement aux membres variables, tels que des champs ou des propriétés.

ID d'erreur : BC32010

Pour corriger cette erreur

1. Assurez-vous que le nom de l'attribut et le nom du membre sont orthographiés correctement.
2. Supprimez l'argument du bloc d'attributs si le membre n'est pas une variable.

Voir aussi

Concepts

Application des attributs

'<nommodule>' est un module et ne peut pas être référencé en tant qu'assembly

'<modulename>' is a module and cannot be referenced as an assembly

Une tentative d'utilisation d'un module en tant qu'assembly a été effectuée.

ID d'erreur : BC31076

Pour corriger cette erreur

- Modifiez la référence à un assembly.

Voir aussi

Référence

Module, instruction
Assembly

'<modificateur>' n'est pas valide dans une déclaration Interface

'<modifier>' is not valid on an Interface declaration

Vous avez utilisé un modificateur non valide dans une déclaration **Interface**. Les seuls modificateurs valides pour les instructions **Sub**, **Function** ou **Property** déclarées dans une déclaration **Interface** sont les mots clés **Overloads** et **Default**. Les autres modificateurs (par exemple, **Public**, **Private**, **Friend**, **Protected**, **Shared**, **Static**, **Overrides**, **MustOverride** ou **Overridable**) ne sont pas valides.

ID d'erreur : BC30397

Pour corriger cette erreur

- Supprimez le modificateur.

Voir aussi

Concepts

Définition des interfaces

'<modificateur>' n'est pas valide dans une déclaration event

'<modifier>' is not valid on an event declaration

Une instruction **Event** contient un mot clé non valide, tel que **ReadOnly**.

ID d'erreur : BC30243

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Event**.

Voir aussi

Référence

Event, instruction

'<modificateur>' n'est pas valide dans une déclaration de méthode

'<modifier>' is not valid on a method declaration

Une instruction **Function** ou **Sub** contient un mot clé non valide.

ID d'erreur : BC30242

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Function** ou **Sub**.

Voir aussi

Référence

Function, instruction (Visual Basic)
Sub, instruction (Visual Basic)

'<modificateur>' n'est pas valide dans une déclaration de variable locale

'<modifier>' is not valid on a local variable declaration

Une instruction **Dim** à l'intérieur d'une procédure contient un mot clé non valide, tel que **Overloads**.

ID d'erreur : BC30247

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Dim**.

Voir aussi

Référence

Dim, instruction (Visual Basic)

'<modificateur>' n'est pas valide dans une déclaration de constante locale

'<modifier>' is not valid on a local constant declaration

Une instruction **Const** à l'intérieur d'une procédure contient un mot clé non valide, tel que **Overloads**.

ID d'erreur : BC30246

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Const**.

Voir aussi

Référence

Const, instruction (Visual Basic)

'<modificateur>' n'est pas valide dans une déclaration Delegate

'<modifier>' is not valid on a Delegate declaration

Vous avez tenté d'utiliser un modificateur, tel que **Shared**, dans une déclaration delegate. Les délégués sont des objets utilisés pour appeler les méthodes d'autres objets, en définissant un constructeur auquel les caractéristiques d'une méthode objet sont passées. La spécification d'un modificateur n'est pas valide dans une déclaration delegate.

ID d'erreur : BC30385

Pour corriger cette erreur

- Supprimez le modificateur.

Voir aussi

Tâches

Comment : appeler une méthode déléguée

Référence

Delegate, instruction

'<modificateur>' n'est pas valide dans un Declare

'<modifier>' is not valid on a Declare

Une instruction **Declare** contient un mot clé non valide, tel que **ReadOnly**.

ID d'erreur : BC30244

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Declare**.

Voir aussi

Référence

Declare, instruction

'<nomméthode>' a plusieurs définitions comportant des signatures identiques

'<methodname>' has multiple definitions with identical signatures

Une déclaration de procédure **Function** ou **Sub** utilise un nom de procédure et une liste d'arguments identiques à ceux d'une déclaration précédente. Une des causes possibles est une tentative de surcharge de la procédure d'origine. Les procédures surchargées doivent avoir des listes d'arguments différentes.

ID d'erreur : BC30269

Pour corriger cette erreur

- Modifiez le nom de la procédure ou la liste d'arguments ou supprimez la déclaration dupliquée.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom
Considérations sur les surcharges de procédures

'<nomméthode>' ne peut pas masquer une méthode déclarée 'MustOverride'

'<methodname>' cannot shadow a method declared 'MustOverride'

Une propriété ou une méthode avec le modificateur **MustOverride** et le même nom est déclarée dans une classe dérivée.

ID d'erreur : BC31404

Pour corriger cette erreur

1. Ajoutez le modificateur **Overrides** à la propriété ou à la méthode de substitution dans la classe dérivée.
2. Supprimez le modificateur **MustOverride** de la propriété ou de la méthode dans la classe de base.

Voir aussi

Référence

MustOverride

'<nomméthode>' ne peut pas se substituer à une méthode qui a été masquée

'<methodname>' cannot override a method that has been shadowed

Une méthode a tenté de se substituer à une méthode qui masque un autre membre.

ID d'erreur : BC31406

Pour corriger cette erreur

- Utilisez le modificateur **Shadows** pour substituer les membres masqués.

Voir aussi

Référence

Shadows
Overrides

'<nomméthode>' ne peut pas être implémenté plus d'une fois

'<methodname>' cannot be implemented more than once

Vous avez tenté d'implémenter une méthode qui a déjà été implémentée.

ID d'erreur : BC30583

Pour corriger cette erreur

1. Vérifiez que le nom de la méthode est correctement orthographié.
2. Si la méthode est redondante, supprimez la déclaration supplémentaire.

Voir aussi

Tâches

Comment : exécuter des actions avec les méthodes

'<nomméthode>' et '<nomméthode>' ne peuvent pas se surcharger mutuellement, car 'ReadOnly' ou 'WriteOnly' les différencie

<methodname>' and '<methodname>' cannot overload each because they differ by 'ReadOnly' or 'WriteOnly'

Vous avez tenté de surcharger deux méthodes qui se distinguent uniquement dans leur déclaration **ReadOnly** et **WriteOnly**. Vous ne pouvez pas utiliser autre chose que la liste des arguments pour différencier les deux versions.

ID d'erreur : BC30366

Pour corriger cette erreur

- Vérifiez que les méthodes ne se distinguent pas uniquement par **ReadOnly** et **WriteOnly**.

Voir aussi

Concepts

Considérations sur les surcharges de procédures

'<méthode1>' ne peut pas se substituer à '<méthode2>', car seuls les types de retour les différencient

'<method1>' cannot override '<method2>' because they differ by their return types

Vous avez tenté de substituer une méthode par une autre méthode qui diffère par son type de retour. Un type peut se substituer à une méthode substituable héritée en déclarant celle-ci avec les mêmes nom et signature et en marquant la déclaration avec le modificateur **Overrides**. Les signatures des deux méthodes doivent correspondre.

ID d'erreur : BC30437

Pour corriger cette erreur

- Vérifiez les types de retour des deux méthodes et modifiez-les, le cas échéant, afin de les faire correspondre.

Voir aussi

Concepts

Substitution de propriétés et de méthodes

'<méthode1>' ne peut pas se substituer à '<méthode2>', car seules les valeurs par défaut des paramètres optionnels les différencient

'<method1>' cannot override '<method2>' because they differ by the default values of optional parameters

Vous avez tenté de substituer une méthode par une autre méthode dont les valeurs par défaut des paramètres optionnels la différencient de la première, ce qui signifie que leurs signatures sont différentes. Un type peut se substituer à une méthode substituable héritée en déclarant celle-ci avec les mêmes nom et signature et en marquant la déclaration avec le modificateur **Overrides**.

ID d'erreur : BC30307

Pour corriger cette erreur

- Vérifiez que les deux méthodes ont la même signature.

Voir aussi

Concepts

Substitution de propriétés et de méthodes
Modificateurs Override

'<méthode1>' ne peut pas se substituer à '<méthode2>', car seuls les paramètres optionnels les différencient

'<method1>' cannot override '<method2>' because they differ by optional parameters

Vous avez tenté de substituer une méthode par une autre méthode dont les valeurs des paramètres optionnels la différencient de la première, ce qui signifie que leurs signatures sont différentes. Un type peut se substituer à une méthode substituable héritée en déclarant celle-ci avec les mêmes nom et signature et en marquant la déclaration avec le modificateur **Overrides**.

ID d'erreur : BC30308

Pour corriger cette erreur

- Vérifiez que les deux méthodes ont la même signature.

Voir aussi

Référence

Overrides

Concepts

Substitution de propriétés et de méthodes

'<méthode1>' ne peut pas se substituer à '<méthode2>', car un paramètre marqué comme 'ByRef' pour l'une et 'ByVal' pour l'autre les différencie

'<method1>' cannot override '<method2>' because they differ by a parameter that is marked as 'ByRef' versus 'ByVal'

Vous avez tenté de substituer une méthode par une autre méthode qui diffère par un paramètre marqué comme **ByRef** au lieu de **ByVal**. Les membres substitués doivent avoir les mêmes arguments que les membres hérités de la classe de base.

ID d'erreur : BC30398

Pour corriger cette erreur

- Vérifiez que les deux paramètres sont **ByRef** ou **ByVal**.

Voir aussi

Concepts

Substitution de propriétés et de méthodes
Passage d'arguments par valeur et par référence

'<méthode1>' ne peut pas se substituer à '<méthode2>' car c'est une instruction 'Declare'

'<method1>' cannot override '<method2>' because it is a 'Declare' statement

Vous avez tenté de substituer un délégué dans le nom de la classe de base déclaré avec une instruction **Declare**.

ID d'erreur : BC30474

Pour corriger cette erreur

1. Modifiez le membre substitué afin qu'il ne soit pas une instruction **Declare**.
2. Ne tentez pas de substituer cette méthode.

Voir aussi

Référence

Declare, instruction

Concepts

Substitution de propriétés et de méthodes

'<méthode1>' ne peut pas se substituer à '<méthode2>' car elle étend l'accessibilité de la méthode de base

'<method1>' cannot override '<method2>' because it expands the access of the base method

Une procédure spécifie **Overrides**, mais déclare une accessibilité moins restrictive que celle de la méthode à substituer. Vous ne pouvez pas étendre l'accessibilité, ce qui signifie que vous ne pouvez pas rendre la méthode de substitution plus accessible que la méthode à laquelle elle se substitue. Par exemple, si la méthode de la classe de base est **Protected**, vous ne pouvez pas la substituer avec une méthode **Public**.

ID d'erreur : BC32203

Pour corriger cette erreur

- Supprimez le mot clé **Overrides** ou modifiez l'accessibilité de sorte qu'elle ait au moins le même niveau de restriction que celle de la méthode de la classe de base.

Voir aussi

Concepts

Substitution de propriétés et de méthodes
Niveaux d'accès dans Visual Basic
Occultation dans Visual Basic

'<méthode1>' et '<méthode2>' ne peuvent pas se surcharger mutuellement car seules les valeurs par défaut des paramètres optionnels les différencient

'<method1>' and '<method2>' cannot overload each other because they differ only by the default values of optional parameters

Vous avez tenté de surcharger une méthode avec une autre méthode qui diffère de la première uniquement dans ses paramètres optionnels. Une méthode avec un paramètre optionnel équivaut à deux méthodes surchargées, l'une avec le paramètre optionnel, l'autre sans celui-ci. Par conséquent, vous ne pouvez pas surcharger une méthode avec une liste d'arguments correspondant à l'une ou l'autre.

ID d'erreur : BC30305

Pour corriger cette erreur

- Vérifiez que les méthodes ne se distinguent pas uniquement par les paramètres optionnels.

Voir aussi

Concepts

Surcharge de procédure

Considérations sur les surcharges de procédures

'<méthode1>' et '<méthode2>' ne peuvent pas se surcharger mutuellement, car seuls les types de retour les différencient

'<method1>' and '<method2>' cannot overload each other because they differ only by return types

Vous avez tenté de surcharger une méthode avec une autre méthode qui diffère de la première uniquement dans son type de retour. Dans une surcharge, vous devez distinguer deux versions quelconques d'une méthode par leurs listes d'arguments ; vous ne pouvez pas utiliser d'autres éléments que ces listes d'arguments pour différencier les méthodes.

ID d'erreur : BC30301

Pour corriger cette erreur

- Vérifiez que les méthodes se distinguent par leurs listes d'arguments.

Voir aussi

Concepts

Surcharge de procédure
Considérations sur les surcharges de procédures

'<méthode1>' et '<méthode2>' ne peuvent pas se surcharger mutuellement, car seuls les paramètres déclarés 'ParamArray' les différencient

'<method1>' and '<method2>' cannot overload each other because they differ only by parameters declared 'ParamArray'

Vous avez tenté de surcharger deux méthodes qui se distinguent uniquement par un ou plusieurs paramètres **ParamArray**. Le compilateur considère qu'une procédure comportant un paramètre **ParamArray** possède un nombre infini de surcharges, qui se différencient par les valeurs passées au tableau de paramètres.

ID d'erreur : BC30368

Pour corriger cette erreur

- Vérifiez que les méthodes ne se distinguent pas uniquement par les paramètres **ParamArray**.

Voir aussi

Concepts

Considérations sur les surcharges de procédures
Tableaux de paramètres

'<méthode1>' et '<méthode2>' ne peuvent pas se surcharger mutuellement, car seuls les paramètres déclarés 'ByRef' ou 'ByVal' les différencient

'<method1>' and '<method2>' cannot overload each other because they differ only by parameters declared 'ByRef' or 'ByVal'

Vous avez tenté de surcharger une méthode avec une autre méthode qui diffère de la première uniquement par un paramètre déclaré **ByRef** ou **ByVal**.

ID d'erreur : BC30345

Pour corriger cette erreur

- Vérifiez que les méthodes ne se distinguent pas uniquement par le nom du paramètre **ByRef** ou **ByVal**.

Voir aussi

Concepts

Surcharge de procédure

Considérations sur les surcharges de procédures

'<méthode1>' et '<méthode2>' ne peuvent pas se surcharger mutuellement, car seuls les paramètres optionnels les différencient

'<method1>' and '<method2>' cannot overload each other because they differ only by optional parameters

Vous avez tenté de surcharger une méthode avec une autre méthode qui diffère de la première uniquement dans ses paramètres optionnels. Une méthode avec un paramètre optionnel équivaut à deux méthodes surchargées, l'une avec le paramètre optionnel, l'autre sans celui-ci. Par conséquent, vous ne pouvez pas surcharger une méthode avec une liste d'arguments correspondant à l'une ou l'autre.

ID d'erreur : BC30300

Pour corriger cette erreur

- Vérifiez que les méthodes ne se distinguent pas uniquement par les paramètres optionnels.

Voir aussi

Concepts

Surcharge de procédure

Considérations sur les surcharges de procédures

'<méthode>' n'est pas accessible dans ce contexte, car elle est '<modificateur>'

'<method>' is not accessible in this context because it is '<modifier>'

Vous avez tenté d'accéder à une méthode qui n'est pas accessible dans ce contexte car elle a été déclarée **Private**. Une cause possible de cette erreur est que le compilateur Visual Basic importe tous les membres d'une classe et ne respecte pas la casse ; les noms différenciés uniquement par la casse peuvent donc entrer en conflit.

ID d'erreur : BC30389

Pour corriger cette erreur

- Pensez à déclarer la méthode **Public**.
- Si l'erreur provient d'une collision de noms, ne différenciez pas les noms en conflit uniquement par la casse.

Voir aussi

Référence

Private (Visual Basic)

<méthode> '<nomméthode1>' doit être déclarée 'Overloads', car une autre '<nomméthode2>' est déclarée 'Overloads'

<method> '<methodname1>' must be declared 'Overloads' because another '<methodname2>' is declared 'Overloads'

Une autre propriété ou méthode portant le même nom est déclarée avec le modificateur **Overloads**.

ID d'erreur : BC31409

Pour corriger cette erreur

1. Ajoutez le modificateur **Overloads** à la propriété ou à la méthode.
2. Supprimez le modificateur **Overloads** de toutes les propriétés ou méthodes portant le même nom.

Voir aussi

Référence

Overloads

<message> Cette erreur peut résulter de la combinaison d'une référence de fichier pour '<NomFichier1>' dans le projet '<NomProjet1>' et d'une référence de fichier pour '<NomFichier2>' dans le projet '<NomProjet2>'

Message d'erreur
<message> Cette erreur peut résulter de la combinaison d'une référence de fichier pour '<NomCheminFichier1>' dans le projet '<NomProjet1>' et d'une référence de fichier pour '<NomCheminFichier2>' dans le projet '<NomProjet2>' Si les deux assemblys sont identiques, essayez de remplacer ces deux références pour qu'elles se situent dans le même emplacement. <message> This error could also be due to mixing a file reference to '<filepathname1>' in project '<projectname1>' with a file reference to '<filepathname2>' in project '<projectname2>'. If both assemblies are identical, try replacing these references so both references are from the same location.

Le code de votre projet accède à un membre d'un autre projet, mais la configuration de votre solution n'autorise pas le compilateur Visual Basic à résoudre la référence.

Pour accéder à un type défini dans un autre assembly, le compilateur Visual Basic doit avoir une référence à cet assembly. Il doit s'agir d'une référence unique et non équivoque qui ne provoque pas de références circulaires parmi des projets.

ID d'erreur : BC30970

Pour corriger cette erreur

1. Déterminez quel projet produit le meilleur assembly pour votre projet à référencer. Pour prendre cette décision, vous pouvez utiliser des critères, tels que la facilité d'accès au fichier et la fréquence des mises à jour.

2. Dans vos propriétés de projet, ajoutez une référence au fichier contenant l'assembly qui définit le type que vous utilisez.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Comment : modifier des propriétés de projet et des paramètres de configuration
Dépannage de références rompues
Concepts
Références de projet
Résolution d'une référence lorsque plusieurs variables ont le même nom
Autres ressources
Références aux espaces de noms et aux composants

<message> Cette erreur peut résulter de la combinaison d'une référence de fichier et d'une référence de projet pour l'assembly '<NomAssembly>

Message d'erreur
<message> Cette erreur peut résulter de la combinaison d'une référence de fichier et d'une référence de projet pour l'assembly '<NomAssembly>. Dans ce cas, essayez de remplacer la référence du fichier à '<NomFichierAssembly>' dans le projet '<NomProjet1>' par une référence de projet à '<NomProjet2>.
<message> This error could also be due to mixing a file reference with a project reference to assembly '<assemblyname>. In this case, try replacing the file reference to '<assemblyfilename>' in project '<projectname1>' with a project reference to '<projectname2>'.

Le code de votre projet accède à un membre d'un autre projet, mais la configuration de votre solution n'autorise pas le compilateur Visual Basic à résoudre la référence.

Pour accéder à un type défini dans un autre assembly, le compilateur Visual Basic doit avoir une référence à cet assembly. Il doit s'agir d'une référence unique et non équivoque qui ne provoque pas de références circulaires parmi des projets.

ID d'erreur : BC30971

Pour corriger cette erreur

1. Déterminez quel projet produit le meilleur assembly pour votre projet à référencer. Pour prendre cette décision, vous pouvez utiliser des critères, tels que la facilité d'accès au fichier et la fréquence des mises à jour.
2. Dans les propriétés de votre projet, ajoutez une référence au projet contenant l'assembly qui définit le type que vous utilisez.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Comment : modifier des propriétés de projet et des paramètres de configuration
Dépannage de références rompues
Concepts
Références de projet
Résolution d'une référence lorsque plusieurs variables ont le même nom
Autres ressources
Références aux espaces de noms et aux composants

'<nomméthode1>' ne peut pas se substituer à '<nomméthode2>', car les types des paramètres optionnels les différencient

'<membername1>' cannot override '<membername2>' because they differ by the types of optional parameters

Les membres surchargés doivent avoir des types de données différents pour un ou plusieurs paramètres requis.

ID d'erreur : BC30697

Pour corriger cette erreur

- Définissez au moins un paramètre requis en tant que type de données différent.

Voir aussi

Référence

Overloads

'<nommembre1>' et '<nommembre2>' ne peuvent pas se charger mutuellement, car seuls les types des paramètres optionnels les différencient

'<membername1>' and '<membername2>' cannot overload each other because they differ only by the types of optional parameters

Les membres surchargés doivent avoir des types de données différents pour un ou plusieurs paramètres requis.

ID d'erreur : BC30696

Pour corriger cette erreur

- Définissez au moins un paramètre requis en tant que type de données différent.

Voir aussi

Référence

Overloads

'<NomMembre>' est obsolète : '<MessageErreur>'

'<membername>' is obsolete: '<errormessage>'

Une instruction tente d'accéder à un membre de classe ou de structure qui a été marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme une erreur.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30668

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Assurez-vous que le nom du membre est orthographié correctement dans la référence du code source.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'<nommembre>' n'est pas déclaré ou le module qui le contient n'est pas chargé dans la session de débogage

'<membername>' is not declared or the module containing it is not loaded in the debugging session

Le membre spécifié n'a pas pu être trouvé.

ID d'erreur : BC30699

Pour corriger cette erreur

1. Vérifiez que le membre est correctement défini et orthographié.
2. Accédez à l'un des membres déclarés dans le module. Dans certains cas, l'environnement de débogage ne peut pas localiser les membres car les modules dans lesquels ils sont déclarés n'ont pas encore été chargés.

Voir aussi

Autres ressources

Débogage dans Visual Studio

'<nommembre>' n'est pas déclaré

'<membername>' is not declared

'<*membername*>' n'est pas déclaré. Les fonctionnalités de l'objet **Debug** sont disponibles dans **System.Diagnostics.Debug** dans l'assembly **System**.

Le nom du membre spécifié n'a pas pu être trouvé.

ID d'erreur : BC30816

Pour corriger cette erreur

1. Vérifiez l'orthographe du membre.
2. Utilisez l'instruction **Imports** ou des membres qualifiés complets définis dans d'autres espaces de noms. Par exemple, la fonction **WriteLine** est déclarée dans l'espace de noms **System.Diagnostics.Debug**.

Voir aussi

Autres ressources

Débogage dans Visual Studio

'<nommembre>' est ambigu dans les interfaces héritées '<nominterface1>' et '<nominterface2>'

'<membername>' is ambiguous across the inherited interfaces '<interfacename1>' and '<interfacename2>'

L'interface hérite de deux ou plusieurs membres ayant le même nom à partir de plusieurs interfaces.

ID d'erreur : BC30685

Pour corriger cette erreur

- Effectuez un cast de la valeur sur l'interface de base que vous souhaitez utiliser. Par exemple :

```
Interface Left
    Sub MySub()
End Interface

Interface Right
    Sub MySub()
End Interface

Interface LeftRight
    Inherits Left, Right
End Interface

Module test
    Sub Main()
        Dim x As LeftRight
        ' x.MySub()  'x is ambiguous.
        CType(x, Left).MySub() ' Cast to base type.
        CType(x, Right).MySub() ' Call the other base type.
    End Sub
End Module
```

Voir aussi

Concepts

Vue d'ensemble des interfaces

'<nommembre>' est déjà déclaré par '<nomprocédure>', qui a été généré pour ce '<nomconteneur>'

'<membername>' is already declared by '<procedurename>', which was generated for this <containername>

Vous avez déclaré un membre ayant le même nom qu'un membre déclaré par le Common Language Runtime.

ID d'erreur : BC30733

Pour corriger cette erreur

- Modifiez le nom du membre afin de supprimer le conflit.

Voir aussi

Autres ressources

Débogage dans Visual Studio

'<nommembre>' existe dans plusieurs interfaces de base

Message d'erreur
'<nommembre>' existe dans plusieurs interfaces de base. Utilisez le nom de l'interface qui déclare '<nommembre>' dans la clause 'Implements' à la place du nom de l'interface dérivée.
'<membername>' exists in multiple base interfaces. Use the name of the interface that declares '<membername>' in the 'Implements' clause instead of the name of the derived interface.

Cette interface hérite des membres portant le même nom à partir de plusieurs interfaces, ce qui crée une ambiguïté.

ID d'erreur : BC31040

Pour corriger cette erreur

- Utilisez le nom de l'interface de définition dans les clauses **Implements** à la place du nom de l'interface dérivée.

Voir aussi

Référence
Implements (Visual Basic)
Autres ressources
Interfaces en Visual Basic

'<NomMembre>' ne peut pas implémenter '<NomInterface>.<NomMembreInterface>', car les contraintes de paramètre de type les différencient

'<membername>' cannot implement '<interfacename>.<interfacemembername>' because they differ by type parameter constraints

Un événement, une propriété ou une procédure générique tente d'implémenter un membre semblable défini dans une interface, mais ils présentent des listes de contraintes différentes sur leurs paramètres de type.

Pour implémenter un membre d'interface, le membre assurant l'implémentation doit correspondre à la signature complète du membre d'interface, mais également au mécanisme de passage de chaque paramètre.

Pour implémenter un membre d'interface générique, le membre qui assure l'implémentation doit également correspondre au nombre de paramètres de type et à la liste de contraintes de chacun d'eux.

Pour plus d'informations sur l'implémentation d'une interface, consultez Implements, mot clé et instruction.

ID d'erreur : BC32078

Pour corriger cette erreur

- Si vous comptez implémenter le membre d'interface, modifiez les contraintes de paramètres de type pour correspondre exactement à celles du membre d'interface.
- Si les contraintes de paramètres de type doivent rester telles quelles, vous ne pouvez pas implémenter le membre d'interface dans cette déclaration. Supprimez le mot clé Implements (Visual Basic) de la déclaration.

Voir aussi

Concepts

Types génériques en Visual Basic
Exemples d'implémentation d'interface dans Visual Basic

'<nommembre>' ne peut pas être déclaré 'Shadows' en dehors d'une classe, d'une structure ou d'une interface

'<membername>' cannot be declared 'Shadows' outside of a class, structure, or interface

Le mot clé **Shadows** s'affiche dans une déclaration au niveau de l'espace de noms, du module ou du fichier. L'occultation est significative uniquement à l'intérieur d'un élément de programmation qui peut hériter d'un élément de base.

ID d'erreur : BC32200

Pour corriger cette erreur

- Supprimez le mot clé **Shadows** ou déplacez la déclaration à l'intérieur d'une classe, d'une structure ou d'une interface.

Voir aussi

Référence

Shadows

Concepts

Occultation dans Visual Basic

'<membre>', déclaré implicitement pour '<nomévénement>', ne peut pas masquer une méthode 'MustOverride' dans la <classe> '<nomclasse>' de base

'<member>', implicitly defined for '<eventname>', cannot shadow a 'MustOverride' method in the base <class> '<classname>'

L'événement spécifié déclare implicitement un membre avec le même nom qu'une méthode déclarée avec le modificateur **MustOverride**.

ID d'erreur : BC31413

Pour corriger cette erreur

- Supprimez le modificateur **MustOverride** de la méthode dans la classe de base ou donnez un nom unique à la propriété ou à la méthode.

Voir aussi

Référence

MustOverride

Autres ressources

Événements en Visual Basic

'<membre>' est déjà déclaré dans cette structure

'<member>' is already declared in this structure

Un membre de l'énumération a été déclaré plusieurs temps dans cette structure.

ID d'erreur : BC31421

Pour corriger cette erreur

- Supprimez la déclaration superflue.

Voir aussi

Autres ressources

Présentation du débogueur

'<membre>' est en conflit avec le membre réservé par ce nom qui est implicitement déclaré dans tous les enums

'<member>' conflicts with the reserved member by this name that is implicitly declared in all enums

Le nom d'un membre de type est en conflit avec le nom d'un membre implicitement déclaré dans toutes les énumérations.

Le compilateur Visual Basic crée des membres implicites correspondant à certains éléments de programmation que vous déclarez. Les énumérations déclarent implicitement le membre `value__member`.

ID d'erreur : BC31420

Pour corriger cette erreur

- Modifiez le nom du membre.

Voir aussi

Concepts

Instructions de déclaration en Visual Basic
Vue d'ensemble des énumérations
Noms d'éléments déclarés

'<fonctionmath1>' n'est pas déclaré

Message d'erreur
'<fonctionmath1>' n'est pas déclaré Cette fonction a été déplacée dans la classe System.Math et s'appelle désormais '<fonctionmath2>'.
'<mathfunction1>' is not declared. Function has moved to the System.Math class and is now called '<mathfunction2>'.

Plusieurs fonctions qui étaient intrinsèques à Visual Basic dans les versions antérieures ont été déplacées dans l'espace de noms System.Math. En conséquence, leurs fonctionnalités sont plus accessibles aux langages de programmation.

ID d'erreur : BC30819

Pour corriger cette erreur

- Utilisez les méthodes déclarées dans **System.Math**.

Voir aussi

Référence
Math
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

'<lsetstmt>' n'est pas déclaré

Message d'erreur
'<lsetstmt>' n'est pas déclaré Les instructions LSet ne sont plus prises en charge ; utilisez Microsoft.VisualBasic.LSet à la place.
'<lsetstmt>' is not declared. LSet statements are no longer supported; use Microsoft.VisualBasic.LSet instead.

Plusieurs fonctions qui étaient intrinsèques à Visual Basic dans les versions antérieures ont été déplacées dans l'espace de noms Microsoft.VisualBasic. En conséquence, leurs fonctionnalités sont plus accessibles aux langages de programmation.

ID d'erreur : BC30820

Pour corriger cette erreur

- Utilisez à la place la fonction **LSet** dans l'espace de noms **Microsoft.VisualBasic**.

Voir aussi

Référence
LSet, fonction
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

'<motclé>' est valide uniquement dans une méthode d'instance

'<keyword>' is valid only within an instance method

Les mots clés **Me**, **MyClass** et **MyBase** font référence à des instances de classe spécifiques. Vous ne pouvez pas les utiliser à l'intérieur d'une procédure **Function** ou **Sub** partagée.

ID d'erreur : BC30043

Pour corriger cette erreur

- Supprimez le mot clé de la procédure ou supprimez le mot clé **Shared** de la déclaration de procédure.

Voir aussi

Référence

Me
MyClass
MyBase

Concepts

Éléments fondamentaux de l'héritage
Assignation des variables objets

'<motclé>' valide uniquement à l'intérieur d'une classe

'<keyword>' is valid only within a class

Un mot clé en relation avec des classes, tel que **Me** ou **MyClass**, est utilisé à l'extérieur d'une définition de classe.

ID d'erreur : BC32002

Pour corriger cette erreur

- Si le code comprenant le mot clé implique des instances de classe, déplacez-le vers une implémentation de classe.
- Si le code comprenant le mot clé ne s'applique pas aux classes, supprimez le mot clé invalide.

Voir aussi

Référence

Me
MyClass
Class, instruction (Visual Basic)

'<motclé>' n'est pas valide dans une structure

'<keyword>' is not valid within a structure

Les structures sont des types valeur, non des types référence. Une structure n'est pas une instance créée à partir d'une classe ; par conséquent, les mots clés **Me**, **MyClass** et **MyBase** n'ont pas de sens dans une structure.

ID d'erreur : BC30044

Pour corriger cette erreur

- Remplacez la structure par une classe ou supprimez le mot clé de la procédure.

Voir aussi

Référence

Me
MyClass
MyBase

Concepts

Éléments fondamentaux de l'héritage

'<motclé>' n'est pas valide dans un module

'<keyword>' is not valid within a Module

Un mot clé en relation avec des instances de classe, tel que **Me** ou **MyBase**, est utilisé à l'intérieur d'un module. Les modules n'ont pas d'héritage ou d'instances.

ID d'erreur : BC32001

Pour corriger cette erreur

- Si le code comprenant le mot clé implique des instances de classe, déplacez-le vers une implémentation de classe.
- Si le code comprenant le mot clé s'applique au module, supprimez le mot clé invalide.

Voir aussi

Référence

Me
MyBase

L'accesseur '<MotClé>' de '<NomPropriété>' est obsolète : '<MessageErreur>' (avertissement Visual Basic)

'<keyword>' accessor of '<propertyname>' is obsolete: '<errormessage>' (Visual Basic Warning)

Une instruction essaie de lire ou d'écrire dans une propriété pour laquelle la procédure correspondante a été marquée avec l'attribut ObsoleteAttribute et la directive pour la traiter comme un avertissement.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40019

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Assurez-vous que le nom de la propriété est orthographié correctement dans la référence du code source.
3. Évitez d'accéder à la propriété en utilisant la méthode (lecture ou écriture) qui a généré ce message.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs
Procédures Property

L'accesseur '<MotClé>' de '<NomPropriété>' est obsolète : '<MessageErreur>' (erreur Visual Basic)

'<keyword>' accessor of '<propertyname>' is obsolete: '<errormessage>' (Visual Basic Error)

Une instruction essaie de lire ou d'écrire une propriété pour laquelle la procédure correspondante a été marquée avec l'attribut ObsoleteAttribute et la directive de la traiter comme une erreur.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30911

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Assurez-vous que le nom de la propriété est orthographié correctement dans la référence du code source.
3. Évitez d'accéder à la propriété via la méthode (lecture ou écriture) qui a généré ce message.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs
Procédures Property

L'accesseur '<MotClé>' de '<NomPropriété>' est obsolète (avertissement Visual Basic)

'<keyword>' accessor of '<propertyname>' is obsolete (Visual Basic Warning)

Une instruction essaie de lire ou d'écrire une propriété pour laquelle la procédure correspondante a été marquée avec l'attribut ObsoleteAttribute et la directive pour la traiter comme un avertissement.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40020

Pour corriger cette erreur

1. Assurez-vous que le nom de la propriété est orthographié correctement dans la référence du code source.
2. Évitez d'accéder à la propriété en utilisant la méthode (lecture ou écriture) qui a généré ce message.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs
Procédures Property

L'accesseur '<MotClé>' de '<NomPropriété>' est obsolète (erreur Visual Basic)

'<keyword>' accessor of '<propertyname>' is obsolete (Visual Basic Error)

Une instruction essaie de lire ou d'écrire une propriété pour laquelle la procédure correspondante a été marquée avec l'attribut ObsoleteAttribute et la directive de la traiter comme une erreur.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30912

Pour corriger cette erreur

1. Assurez-vous que le nom de la propriété est orthographié correctement dans la référence du code source.
2. Évitez d'accéder à la propriété via la méthode (lecture ou écriture) qui a généré ce message.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs
Procédures Property

'<nominterface1>' ne peut pas implémenter '<nomméthode>' car il n'existe pas de <méthode> correspondante dans l'interface '<nominterface2>'

'<interfacename1>' cannot implement '<methodname>' because there is no matching <method> on interface '<interfacename2>'

Vous avez essayé d'implémenter une méthode dans une interface alors qu'il n'existe aucune méthode dans l'interface de laquelle est dérivée la première interface.

ID d'erreur : BC30401

Pour corriger cette erreur

- Assurez-vous que vous avez correctement orthographié le nom de la méthode ou précisez la méthode manquante.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

'<NomInterface>.<NomMembre>' est déjà implémenté par la classe de base '<NomClasseDeBase>'. Réimplémentation de <type> attendue

'<interfacename>.<membername>' is already implemented by the base class '<baseclassname>'. Re-implementation of <type> assumed

Une propriété, une procédure ou un événement dans une classe dérivée utilise une clause **Implements** qui spécifie un membre d'interface qui est déjà implémenté dans la classe de base.

Une classe dérivée peut implémenter de nouveau un membre d'interface qui est implémenté par sa classe de base. La substitution de l'implémentation de la classe de base est une procédure différente. Pour plus d'informations, consultez Implements (Visual Basic).

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42015

Pour corriger cette erreur

- Si vous comptez implémenter de nouveau le membre d'interface, aucune mesure n'est nécessaire. Le code de votre classe dérivée accède au membre implémenté de nouveau à moins que vous utilisiez le mot clé MyBase pour accéder à l'implémentation de la classe de base.
- Si vous ne comptez pas à implémenter de nouveau le membre d'interface, supprimez la clause **Implements** de la propriété, la procédure ou la déclaration de l'événement.

Voir aussi

Concepts

Implements, mot clé et instruction

Autres ressources

Interfaces en Visual Basic

'<clauseimplements>' ne peut pas implémenter '<nomtype>', car '<nomtype>' est un nom réservé

'<implementsclause>' cannot implement '<typename>' because '<typename>' is a reserved name

Une instruction **Implements** a spécifié un nom réservé.

ID d'erreur : BC31415

Pour corriger cette erreur

- Renommez le membre d'interface qui n'a pas pu être implémenté.

Voir aussi

Référence

Implements, instruction

'<nomfonction>' n'est pas déclaré (erreur Visual Basic)

Message d'erreur
'<nomfonction>' n'est pas déclaré. La fonctionnalité d'E/S de fichier est disponible dans l'espace de noms Microsoft.VisualBasic.
'<functionname>' is not declared. File I/O functionality is available in the Microsoft.VisualBasic namespace.

ID d'erreur : BC30815

Pour corriger cette erreur

- Utilisez les fonctions d'E/S sur fichier définies dans **My.Computer.FileSystem**, l'espace de noms **System.IO**, les fonctions runtime Visual Basic ou l'objet **Microsoft.VisualBasic.FileSystem**.

Voir aussi

Référence
My.Computer.FileSystem, objet
Autres ressources
Accès au fichier avec Visual Basic

'<nomfonction>' n'est pas déclaré (erreur du compilateur Visual Basic/Smart Device)

'<functionname>' is not declared (Smart Device/Visual Basic Compiler Error)

<*functionname*> n'est pas déclaré. La fonctionnalité d'E/S de fichier est normalement disponible dans l'espace de noms **Microsoft.VisualBasic**, mais la version ciblée du .NET Compact Framework ne la prend pas en charge.

ID d'erreur : BC30766

Pour corriger cette erreur

- Effectuez des opérations sur des fichiers à l'aide de fonctions définies dans l'espace de noms **System.IO**.

Voir aussi

Référence

System.IO

Autres ressources

Accès au fichier avec Visual Basic

'<fonction>' n'est pas déclaré

Message d'erreur
'<fonction>' n'est pas déclaré. Cette fonction a été déplacée dans l'espace de noms Microsoft.VisualBasic.
'<function>' is not declared. Function has moved to the Microsoft.VisualBasic namespace.

ID d'erreur : BC30818

Pour corriger cette erreur

- Utilisez la fonction de l'espace de noms **Microsoft.VisualBasic** appropriée.

Voir aussi

Référence
Membres de la bibliothèque runtime Visual Basic

Impossible de référencer '<nomfichier>', car il ne s'agit pas d'un assembly

'<filename>' cannot be referenced because it is not an assembly

Vous avez fait référence à un fichier qui n'est pas un assembly .NET

ID d'erreur : BC31519

Pour corriger cette erreur

1. Vérifiez le nom et l'emplacement de l'assembly.
2. Vérifiez que l'assembly n'est pas endommagé ou corrompu.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Concepts

Références de projet

'<expression>' ne peut pas être utilisé en tant que contrainte de type

'<expression>' cannot be used as a type constraint

Une liste de contraintes contient une expression qui ne représente pas une contrainte valide sur un paramètre de type.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Vous pouvez spécifier les éléments requis suivants selon n'importe quelle combinaison :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit hériter d'une classe au plus
- L'argument de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder (inclure la contrainte **New**)

Si la liste des contraintes ne contient pas de classes ou d'interfaces spécifiques, vous pouvez imposer une exigence plus générale en spécifiant l'un des éléments suivants :

- L'argument de type doit être un type valeur (inclure la contrainte **Structure**)
- L'argument de type doit être un type référence (inclure la contrainte **Class**)

Vous ne pouvez pas spécifier **Structure** et **Class** pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32061

Pour corriger cette erreur

- Vérifiez que l'expression et ses éléments sont correctement orthographiés.
- Si l'expression ne répond pas à la précédente liste d'exigences, supprimez-la de la liste des contraintes.
- Si l'expression fait référence à une interface ou une classe, vérifiez que le compilateur a accès à cette interface ou classe. Vous devrez peut-être qualifier son nom et ajouter une référence à votre projet. Pour plus d'informations, consultez « Références aux projets » dans Résolution d'une référence lorsque plusieurs variables ont le même nom.

Voir aussi

Tâches

Comment : qualifier un nom d'élément déclaré
Comment : ajouter ou supprimer des références en Visual Studio

Concepts

Types génériques en Visual Basic
Types valeur et types référence

'<nomévénement>' n'est pas un événement de '<nomconteneur>'

'<eventname>' is not an event of '<containername>'

L'événement spécifié n'est pas déclaré pour l'objet.

ID d'erreur : BC30676

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom de l'événement.
2. Vérifiez que vous accédez à l'objet correct. Utilisez des références qualifiées complètes à l'objet ou l'instruction **Imports** pour importer l'espace de noms approprié si nécessaire.

Voir aussi

Référence

Imports, instruction

Autres ressources

Événements en Visual Basic

'<nomévénement>' est un événement. Il ne peut donc pas être appelé directement

'<eventname>' is an event, and cannot be called directly

'*<eventname>*' est un événement et ne peut donc pas être appelé directement. Utilisez une instruction **RaiseEvent** pour déclencher un événement.

Un appel de procédure spécifie un événement pour le nom de la procédure. Un gestionnaire d'événements est une procédure, mais l'événement lui-même est un moyen de signalisation qui doit être déclenché et géré.

ID d'erreur : BC32022

Pour corriger cette erreur

- Utilisez une instruction **RaiseEvent** pour signaler un événement et appeler les procédures qui le gèrent.

Voir aussi

Tâches

Comment : ajouter des événements à une classe

Référence

RaiseEvent, instruction

'<nomévénement>' définit implicitement '<nommembre>', qui est en conflit avec un membre déclaré implicitement dans <type> '<nomtype>'

'<eventname>' implicitly defines '<membername>', which conflicts with a member implicitly declated in <type> '<typename>'

Le nom d'un membre de type est en conflit avec le nom d'un membre implicitement créé pour un événement. Les événements créent implicitement plusieurs variables. Par exemple, la déclaration `Event X` déclare implicitement les noms `XEventHandler`, `XEvent`, `add_X` et `remove_X`.

ID d'erreur : BC31059

Pour corriger cette erreur

- Renommez le membre déclaré explicitement afin de supprimer le conflit de noms.

Voir aussi

Concepts

Instructions de déclaration en Visual Basic

Autres ressources

Événements en Visual Basic

<erreur> : '<nomstructure1>' contient '<nomstructure2>'

<error>: '<structurename1>' contains '<structurename2>'

Une définition de structure circulaire apparaît lorsqu'une structure se désigne comme l'un de ses propres membres.

ID d'erreur : BC30293

Pour corriger cette erreur

- Modifiez le nom du membre de structure.

Voir aussi

Autres ressources

Structures constituées à partir de vos propres types de données

<erreur> : '<nomconstructeur1>' appelle '<nomconstructeur2>'

<error>: '<constructorname1>' calls '<constructorname2>'

Un appel de constructeur circulaire se produit. Un constructeur appelle **Me.New()** ou **MyClass.New()**. Une des causes possibles est une tentative d'appel d'un constructeur surchargé avec une liste d'arguments différente.

ID d'erreur : BC30297

Pour corriger cette erreur

- Utilisez une liste d'arguments différente pour appeler un constructeur surchargé.
- S'il n'y a pas de surcharges accessibles, supprimez l'appel adressé à **Me.New()** ou **MyClass.New()**.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

<erreur> : '<nomclasse1>' hérite de '<nomclasse2>'

<error>: '<classname1>' inherits from '<classname2>'

Une hiérarchie d'héritage circulaire a été détectée. Une classe est désignée comme héritant d'elle-même ou d'une autre classe qui hérite, directement ou au final, de celle-ci.

Ce message peut apparaître plusieurs fois afin de suivre le chemin d'héritage circulaire.

ID d'erreur : BC30256

Pour corriger cette erreur

- Interrompez la circularité en supprimant au moins une instruction **Inherits** dans le chemin d'héritage circulaire.

Voir aussi

Concepts

Éléments fondamentaux de l'héritage

'<constanteempty>' n'est pas déclaré

Message d'erreur
'<constanteempty>' n'est pas déclaré. La constante Empty n'est plus prise en charge ; utilisez Nothing à la place.
'<emptyconstant>' is not declared. Empty constant is no longer supported; use Nothing instead.

Une déclaration ou une instruction d'assignation tente d'assigner une valeur **Empty** à une variable, une constante, un membre d'énumération, une propriété ou un retour de fonction.

Dans les versions antérieures de Visual Basic, le mot clé **Empty** était utilisé pour représenter le stockage des données non initialisées. Visual Basic 2005 ne prend pas en charge **Empty**. Une variable non initialisée contient la valeur par défaut pour son type de données. Pour plus d'informations sur les valeurs par défaut, consultez « Valeurs par défaut » dans Dim, instruction (Visual Basic).

Le mot clé Nothing (Visual Basic) représente la valeur par défaut d'un type de données. Vous pouvez l'utiliser à la place de **Empty**.

ID d'erreur : BC30823

Pour corriger cette erreur

- Utilisez **Nothing** au lieu de **Empty**.

  - ou -

- Utilisez la valeur par défaut appropriée pour le type de données de l'élément de programmation.

  - ou -

- S'il s'agit d'une déclaration de variable, n'assignez pas de valeur initiale. Cela entraîne l'initialisation de la variable à sa valeur par défaut.

Voir aussi

Référence
Nothing (Visual Basic)
Concepts
Récapitulatif des modifications relatives à la prise en charge des éléments de programmation

'<NomÉlément>' fait référence au type '<NomType>' dans le projet '<NomProjet>', mais le type '<NomType>' est introuvable dans le projet '<NomProjet>'

'<elementname>' refers to type '<typename>' in project '<projectname>', but type '<typename>' was not found in project '<projectname>'

Une expression accède à une classe, une structure, un module ou une interface référencé(e) dans un autre projet, mais ce projet ne contient pas le type spécifié.

Cette erreur se produit lorsque votre projet fait une référence indirecte à un autre projet dans la même solution. En général, votre projet fait une référence à un assembly qui fait une référence à l'autre projet. Si l'assembly accède au type spécifié dans l'autre projet, la référence indirecte au type est établie. Toutefois, si le type n'est pas présent dans l'autre projet, cette erreur est générée.

ID d'erreur : BC30960

Pour corriger cette erreur

- Si le type cité n'est plus défini nulle part, supprimez ou remplacez l'instruction qui tente d'y accéder. Vous pouvez également avoir besoin d'apporter la même modification dans l'assembly qui fournit la référence indirecte au type cité.
- Si le type cité est défini quelque part, faites une référence directe au projet ou à l'assembly qui le définit.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Concepts

Références de projet

Autres ressources

Références aux espaces de noms et aux composants

Gestion des références

'<NomÉlément>' est obsolète : '<MessageErreur>'

'<elementname>' is obsolete: '<errormessage>'

Une instruction essaie d'accéder à un élément de programmation qui a été marqué avec l'attribut ObsoleteAttribute et la directive de le traiter comme un avertissement.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40000

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Assurez-vous que le nom de l'élément est orthographié correctement dans la référence du code source.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'<NomÉlément>' est obsolète (avertissement Visual Basic)

'<elementname>' is obsolete (Visual Basic Warning)

Une instruction essaie d'accéder à un élément de programmation qui a été marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme un avertissement.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40008

Pour corriger cette erreur

- Assurez-vous que le nom de l'élément est orthographié correctement dans la référence du code source.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'<NomÉlément>' est obsolète (erreur Visual Basic)

'<elementname>' is obsolete (Visual Basic Error)

Une instruction essaie d'accéder à un élément de programmation qui a été marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme une erreur.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC31075

Pour corriger cette erreur

- Assurez-vous que le nom de l'élément est orthographié correctement dans la référence du code source.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'<NomÉlément>' est ambigu, car plusieurs types de membres portant ce nom existent dans <type> '<NomType>'

'<elementname>' is ambiguous because multiple kinds of members with this name exist in <type> '<typename>'

Une expression accède à un élément de programmation défini dans une classe, une structure, un module ou une interface qui contient plusieurs membres avec le même nom.

La cause la plus probable de cette erreur est le *respect de la casse*. Les noms Visual Basic ne respectent pas la casse, ce qui signifie que vous pouvez les capitaliser différemment à différents endroits dans votre code. Par exemple, si vous définissez une variable avec le nom `XYZ` et que vous y accédez ultérieurement en tant que `xyz`, le compilateur considère les deux noms comme équivalents.

Toutefois, d'autres langages, tels que Visual C# et Visual C++, respectent la casse. Dans ces langages, `XYZ` et `xyz` ne sont pas considérés comme le même nom. Par conséquent, une classe écrite dans ces langages pourrait définir une variable nommée `XYZ` et une propriété nommée `xyz`. Le common language runtime (CLR) maintient le respect de la casse dans les assemblys. Toutefois, si une application Visual Basic accède à un assembly avec les noms `XYZ` et `xyz`, ils apparaissent comme un même nom.

ID d'erreur : BC31429

Pour corriger cette erreur

1. Si vous avez le contrôle sur le code source du type de définition, envisagez de renommer les membres afin qu'ils soient différenciés par d'autres éléments que la casse. Cela n'est peut-être pas possible si le type de définition a déjà été publié et qu'il est utilisé par d'autres applications.
2. Si vous ne pouvez pas renommer les membres dans le type définition, supprimez de votre code l'élément de programmation cité. Vous ne pouvez pas accéder à un élément qui apparaît à Visual Basic comme ayant plusieurs définitions.

Voir aussi

Tâches

Dépannage des variables en Visual Basic

Concepts

Noms d'éléments déclarés

Autres ressources

Common Language Runtime

L'alias '<NomÉlément>' utilisé pour les importations de '<NomÉlémentQualifié>' ne fait pas référence à un Namespace, Class, Structure, Interface, Enum ou Module

'<elementname>' for the Imports alias to '<qualifiedelementname>' does not refer to a Namespace, Class, Structure, Interface, Enum or Module

Une instruction Imports spécifie un élément de programmation qui ne peut pas être importé.

L'instruction **Imports** permet de réduire ou d'éliminer la nécessité de placer une chaîne qualifiante devant un nom d'élément. Vous qualifiez l'élément dans l'instruction **Imports** elle-même pour fournir un chemin sans ambiguïté à une déclaration unique de l'élément. Ensuite, vous n'avez pas besoin de qualifier les références à l'élément.

Imports s'utilise le plus souvent pour les espaces de noms, mais vous pouvez également importer une classe, un module, une structure, une interface ou une énumération pour permettre une référence à ses éléments sans une longue chaîne qualifiante.

Pour plus d'informations, consultez « Importation d'éléments conteneurs » dans
Résolution d'une référence lorsque plusieurs variables ont le même nom.

ID d'erreur : BC30798

Pour corriger cette erreur

1. Vérifiez l'orthographe des éléments dans la chaîne qualifiante de l'instruction **Imports**, surtout le dernier élément de la chaîne, qui constitue l'élément que vous qualifiez.
2. Vérifiez que l'élément que vous qualifiez est d'un type prêt pour l'opération (espace de noms, classe, module, structure, interface ou énumération). Si tel n'est pas le cas, supprimez l'instruction **Imports**.

Voir aussi

Concepts

Références et l'instruction Imports

'<NomTypeDérivé>' ne peut pas hériter de <type> '<NomTypeConstruitDeBase>', car il étend l'accès du type '<NomTypeInterne>' en dehors de l'assembly

'<derivedtypename>' cannot inherit from <type> '<constructedbasetypename>' because it expands the access of type '<internaltypename>' outside the assembly

Une classe ou une interface dérivée essaie d'étendre le niveau d'accès d'un type restreint en l'utilisant comme argument de type dans une classe ou une interface de base.

Le code suivant peut générer ce message d'erreur.

```
Public Class baseClass(Of t)
End Class
Public Class derivedClass
    Inherits baseClass(Of restrictedStructure)
End Class
Friend Structure restrictedStructure
    Dim firstMember As Integer
End Structure
```

Le code à l'extérieur de l'assembly n'est pas autorisé à accéder à `restrictedStructure`. Toutefois, tout code qui peut le référencer peut accéder à `derivedClass`. Par conséquent, `derivedClass` ne peut pas hériter de `baseClass` si `restrictedStructure` est utilisé comme argument de type, car cela peut exposer `restrictedStructure` au code d'un assembly.

ID d'erreur : BC30922

Pour corriger cette erreur

- Modifiez les niveaux d'accès des classes ou des interfaces afin que le type dérivé n'étende pas le niveau d'accès du type restreint.

  - ou -

- Si vous ne pouvez pas modifier les niveaux d'accès, n'utilisez pas le type restreint comme argument de type lors de la construction de la classe ou de l'interface de base.

Voir aussi

Référence

Inherits, instruction
Liste de types

Concepts

Éléments fondamentaux de l'héritage
Niveaux d'accès dans Visual Basic
Types génériques en Visual Basic

'<NomTypeDérivé>' ne peut pas hériter de <type> '<NomTypeConstruitDeBase>', car il étend l'accès du type '<NomTypeInterne>' en <région> '<NomRégion>'

'<derivedtypename>' cannot inherit from <type> '<constructedbasetypename>' because it expands the access of type '<internaltypename>' to <region> '<regionname>'

Une classe ou une interface dérivée essaie d'étendre le niveau d'accès d'un type interne en l'utilisant comme argument de type dans une classe ou une interface de base.

Le code suivant peut générer ce message d'erreur.

```
Public Class containingClass
    Public Class baseClass(Of t)
    End Class
    Friend Class derivedClass
        Inherits baseClass(Of internalStructure)
    End Class
    Private Structure internalStructure
        Dim firstMember As Integer
    End Structure
End Class
```

Le code à l'extérieur de `containingClass` n'est pas autorisé à accéder à `internalStructure`. Toutefois, `derivedClass` est accessible à partir de n'importe quel code dans le même assembly. Par conséquent, `derivedClass` ne peut pas hériter de `baseClass` s'il utilise `internalStructure` comme argument de type, car cela peut exposer `internalStructure` dans l'ensemble de la zone de code de définition.

ID d'erreur : BC30921

Pour corriger cette erreur

- Modifiez les niveaux d'accès des classes ou des interfaces afin que le type dérivé n'étende pas le niveau d'accès du type interne.

  - ou -

- Si vous ne pouvez pas modifier les niveaux d'accès, n'utilisez pas le type interne comme argument de type lors de la construction de la classe ou de l'interface de base.

Voir aussi

Référence

Inherits, instruction
Liste de types

Concepts

Éléments fondamentaux de l'héritage
Niveaux d'accès dans Visual Basic
Types génériques en Visual Basic

'<déclaration1>' ne peut pas se substituer à '<déclaration2>' car elles ont des niveaux d'accès différents

'<declaration1>' cannot override '<declaration2>' because they have different access levels

Une déclaration de procédure ou de propriété tente de se substituer à un élément hérité du même nom, mais elle spécifie une accessibilité différente de celle de l'élément hérité. L'accessibilité de l'élément hérité, telle que **Public** ou **Private**, doit être préservée lors de la substitution.

ID d'erreur : BC30266

Pour corriger cette erreur

- Modifiez l'accessibilité de l'élément de substitution afin qu'il corresponde à celui de l'élément hérité.

Voir aussi

Concepts

Substitution de propriétés et de méthodes
Niveaux d'accès dans Visual Basic

'<déclaration1>' ne peut pas se substituer à '<déclaration2>', car il est déclaré 'Shared'

'<declaration1>' cannot override '<declaration2>' because it is declared 'Shared'

Une déclaration de procédure ou de propriété tente de se substituer à un élément hérité du même nom, mais l'élément hérité est spécifié comme **Shared**. Les éléments partagés ne sont associés à aucune instance de la classe ; par conséquent, ils ne peuvent pas être substitués.

ID d'erreur : BC30268

Pour corriger cette erreur

- Supprimez le mot clé **Shared** de l'élément hérité ou supprimez la déclaration de substitution.

Voir aussi

Concepts

Substitution de propriétés et de méthodes

'<déclaration1>' ne peut pas se substituer à '<déclaration2>', car il est déclaré 'NotOverridable'

'<declaration1>' cannot override '<declaration2>' because it is declared 'NotOverridable'

Une déclaration de procédure ou de propriété tente de se substituer à un élément hérité du même nom, mais l'élément hérité est spécifié comme **NotOverridable**.

ID d'erreur : BC30267

Pour corriger cette erreur

- Supprimez le mot clé **NotOverridable** de la déclaration de l'élément hérité ou supprimez la déclaration de substitution.

Voir aussi

Concepts

Substitution de propriétés et de méthodes

'<nomclasse1>' ne peut pas hériter de <type> '<nomclasse2>', car '<nomclasse2>' est déclaré comme 'NotInheritable'

'<classname1>' cannot inherit from <type> '<classname2>' because '<classname2>' is declared 'NotInheritable'

Une classe tente d'hériter d'une autre classe, mais la classe de base souhaitée est spécifiée comme **NotInheritable**. Les classes **NotInheritable** sont utilisées principalement pour empêcher toute dérivation accidentelle.

ID d'erreur : BC30299

Pour corriger cette erreur

- Supprimez le mot clé **NotInheritable** de la définition de la classe de base souhaitée ou supprimez l'instruction **Inherits**.

Voir aussi

Référence

NotInheritable

Inherits, instruction

Concepts

Éléments fondamentaux de l'héritage

'<NomClasse>' n'est pas conforme CLS, car l'interface '<NomInterface>' qu'il implémente n'est pas conforme CLS

'<classname>' is not CLS-compliant because the interface '<interfacename>' it implements is not CLS-compliant

Une classe ou une interface est marquée comme `<CLSCompliant(True)>` lorsqu'elle dérive ou implémente un type qui est marqué comme `<CLSCompliant(False)>` ou qui n'est pas marqué.

Pour qu'une classe ou une interface soit conforme au Spécification CLS (Common Language Specification) (CLS), l'intégralité de sa hiérarchie d'héritage doit être conforme. Cela signifie que chaque type duquel elle hérite, directement ou indirectement, doit être conforme. De même, si une classe implémente une ou plusieurs interfaces, celles-ci doivent être conformes dans leurs hiérarchies d'héritage.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40029

Pour corriger cette erreur

- Si la conformité CLS est requise, définissez ce type dans une hiérarchie d'héritage ou un modèle d'implémentation différent.
- Si ce type doit demeurer dans sa hiérarchie d'héritage ou son modèle d'implémentation actuel, supprimez **CLSCompliantAttribute** de sa définition ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Concepts

Écriture d'un code conforme CLS

'<NomClasse>' n'est pas conforme CLS, car il dérive de '<NomClasseDeBase>', qui n'est pas conforme CLS

'<classname>' is not CLS-compliant because it derives from '<baseclassname>', which is not CLS-compliant

Une classe ou une interface est marquée comme `<CLSCompliant(True)>` lorsqu'elle dérive d'un type ou en implémente un qui est marqué comme `<CLSCompliant(False)>` ou qui n'est pas marqué.

Pour qu'une classe ou une interface soit conforme Spécification CLS (Common Language Specification) (CLS), toute sa hiérarchie d'héritage doit être conforme. Cela signifie que chaque type dont elle hérite, directement ou indirectement, doit être conforme. De même, si une classe implémente une ou plusieurs interfaces, elles doivent être conformes dans l'ensemble de leurs hiérarchies d'héritage.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40026

Pour corriger cette erreur

- S'il est nécessaire d'avoir une conformité CLS, définissez ce type dans une hiérarchie d'héritage ou dans un modèle d'implémentation différent.
- Si ce type doit rester dans sa hiérarchie d'héritage ou dans son modèle d'implémentation actuel, supprimez l'**CLSCompliantAttribute** de sa définition ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Concepts

Écriture d'un code conforme CLS

'<NomInterfaceBase>.<NomMembre>' de 'implements <NomInterfaceDérivée>' est déjà implémenté par la classe de base '<NomClasseDeBase>'. Réimplémentation de <type> attendue

'<baseinterfacename>.<membername>' from 'implements <derivedinterfacename>' is already implemented by the base class '<baseclassname>'. Re-implementation of <type> assumed

Une propriété, une procédure ou un événement dans une classe dérivée utilise une clause **Implements** qui spécifie un membre d'interface dérivée qui est déjà implémenté sur l'interface de base de la classe de base.

Le membre implémenté est défini par l'interface de base et hérité par l'interface dérivée. La classe de base implémente directement l'interface de base. La classe dérivée implémente l'interface dérivée et peut facilement oublier le fait que la classe de base a déjà implémenté le membre.

Une classe dérivée peut implémenter de nouveau un membre d'interface qui est implémenté par sa classe de base. La substitution de l'implémentation de la classe de base est une procédure différente. Pour plus d'informations, consultez Implements (Visual Basic).

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42014

Pour corriger cette erreur

- Si vous comptez implémenter de nouveau le membre d'interface, aucune mesure n'est nécessaire. Le code de votre classe dérivée accède au membre implémenté de nouveau à moins que vous utilisiez le mot clé MyBase pour accéder à l'implémentation de la classe de base.
- Si vous ne comptez pas à implémenter de nouveau le membre d'interface, supprimez la clause **Implements** de la déclaration de la propriété, de la procédure ou de l'événement.

Voir aussi

Concepts

Implements, mot clé et instruction

Autres ressources

Interfaces en Visual Basic

'<nomattribut>' ne peut pas être appliqué plusieurs fois à un assembly

'<attributename>' cannot be applied more than once to an assembly

L'attribut spécifié ne peut être appliqué qu'une seule fois à un attribut.

ID d'erreur : BC31521

Pour corriger cette erreur

1. Supprimez les applications redondantes de cet attribut.
2. Si vous utilisez un attribut personnalisé que vous avez développé, modifiez **AttributeUsageAttribute** et affectez à la propriété **AllowMultiple** la valeur **True**.

Voir aussi

Référence

AttributeUsageAttribute Class
AttributeUsageAttribute.AllowMultiple Property

'<attribut>' ne peut pas être appliqué, car le format du GUID '<nombre>' n'est pas correct

'<attribute>' cannot be applied because the format of the GUID '<number>' is not correct

Un bloc d'attributs **COMClassAttribute** spécifie un GUID (Identificateur Global Unique) qui n'est pas conforme au format correct. **COMClassAttribute** utilise des GUID pour identifier de manière unique la classe, l'interface et l'événement de création.

Un GUID se compose de 16 octets, dont les huit premiers sont numériques et les huit derniers sont binaires. Il est généré par les utilitaires Microsoft, tels que uuidgen.exe et est unique dans l'espace et le temps.

ID d'erreur : BC32500

Pour corriger cette erreur

1. Déterminez le GUID ou les GUID corrects nécessaires pour identifier l'objet COM.
2. Assurez-vous que les chaînes du GUID présentées au bloc d'attributs **COMClassAttribute** ont été copiées correctement.

Voir aussi

Référence

ComClassAttribute, classe
Guid Structure

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'<nomassembly>' est un assembly et ne peut pas être référencé en tant que module

'<assemblyname>' is an assembly and cannot be referenced as a module

Une tentative d'utilisation d'un assembly en tant que module a été effectuée.

ID d'erreur : BC31077

Pour corriger cette erreur

- Modifiez la référence à un module.

Voir aussi

Référence

Module, instruction

Assembly

'<nomargument>' n'est pas un paramètre de '<déclarationméthode>'

'<argumentname>' is not a parameter of '<methoddeclaration>'

Un appel de procédure fournit un argument supplémentaire non défini par la procédure.

ID d'erreur : BC30272

Pour corriger cette erreur

- Supprimez l'argument superflu de la liste d'arguments.

Voir aussi

Concepts

Paramètres et arguments d'une procédure
Résolution d'une référence lorsque plusieurs variables ont le même nom

'}' attendu

'}' expected

Un initialiseur de tableau ou une liste de contraintes ne s'est pas terminé de façon valide.

Les valeurs d'élément permettant d'initialiser un tableau doivent être mises entre accolades (**{}**).

```
Dim demoArray() As Integer = New Integer() {1, 2, 3}
```

De même, les contraintes d'une liste de contraintes pour un paramètre de type générique doivent être mises entre accolades.

```
Public Class dictionaryMaker(Of t As {IComparable, IDisposable, New})
```

ID d'erreur : BC30370

Pour corriger cette erreur

- Utilisez « } » pour terminer l'initialiseur de tableau ou la liste de contraintes.

Voir aussi

Tâches

Comment : initialiser une variable tableau

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Autres ressources

Tableaux dans Visual Basic

'<NomType>' est un type générique et requiert des arguments de type

'<typename>' is a generic type and requires type arguments

Une variable, un paramètre de procédure ou un retour de fonction est déclaré comme ayant le type d'une classe ou d'une structure générique, mais la déclaration ne fournit pas d'arguments de type.

De par sa nature, chaque classe ou structure générique est définie avec au moins un paramètre de type. Lorsque vous utilisez un type générique pour déclarer une classe ou une structure construite, vous devez fournir un argument de type pour chaque paramètre de type défini par le type générique.

ID d'erreur : BC32076

Pour corriger cette erreur

- Ajoutez une liste de types à la déclaration, entourée par des parenthèses et commençant par le mot clé **Of**.

Voir aussi

Référence

Of

Liste de types

Concepts

Types génériques en Visual Basic

'<constructeur>' dans le type généré par le concepteur '<type>' doit appeler la méthode InitializeComponent

'<constructor>' in designer-generated type '<type>' should call InitializeComponent method

Un constructeur dans un type généré par le concepteur n'appelle pas la méthode `InitializeComponent` du type.

Chaque constructeur dans un type généré par le concepteur doit appeler la méthode `InitializeComponent` du type.

ID d'erreur : BC40054

Pour corriger cette erreur

- Ajoutez un appel à la méthode `InitializeComponent` dans le constructeur.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

DesignerGeneratedAttribute

'<NomType>' dans l'assembly '<NomAssembly>' a été transféré à lui-même et est donc un type non pris en charge

'<typename>' in assembly '<assemblyname>' has been forwarded to itself and so is an unsupported type

Un assembly utilise l'TypeForwardedToAttribute pour transférer un de ses types à un autre assembly, mais il spécifie le même type dans le même assembly.

On entend par *transfert de type* la réassignation de la définition d'une classe, d'une structure, d'une interface, d'un délégué ou d'une énumération à un assembly autre que celui dans lequel il a été initialement défini. Il est souvent utilisé conjointement à la *refactorisation de code* par laquelle vous fractionnez un assembly en deux ou plusieurs assemblys ou déplacez le code d'un assembly à un autre.

Le transfert d'un type à lui-même entraîne un transfert circulaire. Si un autre assembly essayait d'accéder au type transféré, il passerait par un transfert interminable sans jamais parvenir à un type qui n'avait pas été transféré.

ID d'erreur : BC31425

Pour corriger cette erreur

- Transférez le type à un type d'un assembly différent ou ne le transférez pas du tout.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Référence

Type Forwarding

TypeForwardedToAttribute

Concepts

Références de projet

'<NomMembre> porte le même nom qu'un paramètre de type

'<membername> has the same name as a type parameter

Un membre d'une classe ou d'une structure générique porte le même nom qu'un paramètre de type de cette classe ou structure.

Chaque variable, constante, énumération, procédure, propriété, événement et délégué doit porter un nom différent de chaque paramètre de type.

ID d'erreur : BC32054

Pour corriger cette erreur

- Modifiez le nom du membre de la classe ou la structure afin qu'il soit différent du nom de chaque paramètre de type.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

'<nom>' ne peut pas se référer à lui-même via son instance par défaut ; utilisez 'Me' à la place

'<name>' cannot refer to itself through its default instance, use 'Me' instead

Une tentative a été effectuée à partir d'un formulaire pour faire référence à ce formulaire en tant qu'instance par défaut. Cela peut entraîner le formulaire à s'appeler lui-même de manière récursive.

Dans la plupart des cas, vous devez utiliser **Me** lors d'une référence à l'instance actuelle du formulaire, plutôt que d'utiliser l'instance par défaut.

ID d'erreur : BC31139

Pour corriger cette erreur

- Utilisez **Me** pour faire référence à l'objet.

Voir aussi

Autres ressources

Présentation du débogueur

'<NomType>' ne peut pas hériter de <type> '<NomTypeBase>' car il étend l'accès du <type> de base en dehors de l'assembly

'<typename>' cannot inherit from <type> '<basetypename>' because it expands the access of the base <type> outside the assembly

Une classe ou une interface hérite d'une classe ou d'une interface de base qui présente un niveau d'accès moins restrictif.

Par exemple, une interface **Public** hérite d'une interface **Friend**, ou une classe **Protected** hérite d'une classe **Private**. Cela expose la classe ou l'interface de base à un accès au-delà du niveau prévu.

ID d'erreur : BC30910

Pour corriger cette erreur

- Modifiez le niveau d'accès de la classe ou de l'interface dérivée pour qu'il soit au moins aussi restrictif que celui de la classe ou de l'interface de base.

  - ou -

- Si un niveau d'accès moins restrictif doit être utilisé, supprimez l'instruction **Inherits**. Vous ne pouvez pas hériter d'une classe ou d'une interface de base plus restreinte.

Voir aussi

Référence

Class, instruction (Visual Basic)
Interface, instruction (Visual Basic)
Inherits, instruction

Concepts

Niveaux d'accès dans Visual Basic

'<NomMembre>' ne peut pas exposer le type '<NomType>' en dehors du projet via <TypeConteneur> '<NomTypeConteneur>'

'<membername>' cannot expose type '<typename>' outside the project through <containertype> '<containertypename>'

Une variable, un paramètre de procédure ou un retour de fonction est exposé en dehors de son conteneur, mais il est déclaré comme un type qui ne doit pas être exposé en dehors du conteneur.

Le code squelette suivant affiche une situation qui génère cette erreur.

```
Private Class privateClass
End Class
Public Class mainClass
    Public exposedVar As New privateClass
End Class
```

Un type qui est déclaré **Protected**, **Friend**, **Protected Friend** ou **Private** est conçu pour avoir un accès limité en dehors de son contexte de déclaration. Son utilisation comme type de données d'une variable avec un accès moins limité nuit à cet objectif. Dans le code squelette précédent, `exposedVar` est **Public** et expose `privateClass` au code qui ne doit pas y avoir accès.

ID d'erreur : BC30909

Pour corriger cette erreur

- Modifiez le niveau d'accès de la variable, du paramètre de procédure ou du retour de fonction de sorte qu'il soit au moins aussi restrictif que le niveau d'accès de son type de données.

Voir aussi

Concepts

Niveaux d'accès dans Visual Basic

Le type '<NomType>' ne peut pas implémenter l'interface '<NomInterface>', car il déclare le '<SignatureÉvénement>' qui a un type de retour

Type '<typename>' cannot implement interface '<interfacename>' because it declares '<eventsignature>' which has a return type

Une classe ou une structure tente d'implémenter une interface qui déclare un événement qui retourne une valeur.

Visual Basic ne prend pas en charge actuellement la déclaration d'événements qui retournent des valeurs.

ID d'erreur : BC30945

Pour corriger cette erreur

- Supprimez l'instruction **Implements** de la définition de la classe ou de la structure, ou implémentez une autre interface.

Voir aussi

Référence

Implements, instruction

Concepts

Événements et gestionnaires d'événements

Implements, mot clé et instruction

'.' attendu

'.' expected

Le code possède une clause **Handles** qui contient un point d'exclamation (**!**).

ID d'erreur : BC30287

Pour corriger cette erreur

- Si la clause **Handles** fait référence à un événement à l'intérieur d'un objet, utilisez un point (**.**) pour séparer l'objet de l'événement.

  Cet exemple gère l'événement **Click** de l'objet `Button1`.

  VB

```
Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
```

Voir aussi

Référence

Handles

')' attendu

')' expected

Une liste, telle qu'une liste d'arguments dans une définition de procédure ou un appel de procédure, n'est pas terminée par une parenthèse fermante (**)**).

ID d'erreur : BC30198

Pour corriger cette erreur

- Ajoutez la parenthèse fermante à la fin de la liste.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

'(' inattendu

Message d'erreur
'(' inattendu. Les tableaux de types génériques non instanciés ne sont pas autorisés.
'(' unexpected. Arrays of uninstantiated generic types are not allowed.

Visual Basic ne peut pas compiler un tableau à moins qu'il soit d'un type de données spécifique. Vous ne pouvez pas utiliser un paramètre de type de données d'un type générique comme type de données d'un tableau.

ID d'erreur : BC32095

Pour corriger cette erreur

- Si vous devez utiliser un tableau, vous devez le déclarer comme étant d'un type de données spécifique. Vous ne pouvez pas utiliser un paramètre de type de données.
- Si un groupe d'éléments du type de données doit être fourni pour un paramètre de type de données, vous devez utiliser une collection au lieu d'un tableau.

Voir aussi

Concepts
Types génériques en Visual Basic
Collections en Visual Basic
Gestion de groupes d'objets

'(' attendu

'(' expected

Une liste d'arguments dans un appel de procédure ne commence pas par une parenthèse ouvrante (**(**).

ID d'erreur : BC30199

Pour corriger cette erreur

- Ajoutez une parenthèse ouvrante au début de la liste.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

L'instruction '#Region' doit se terminer par un '#End Region' correspondant

'#Region' statement must end with a matching '#End Region'

Utilisez la directive **#Region** pour spécifier un bloc de code que vous pouvez développer ou réduire lors de l'utilisation de la fonctionnalité mode Plan de l'éditeur de code de Visual Studio. Le début et la fin des instructions **#Region** doivent figurer dans le même bloc de code.

ID d'erreur : BC30681

Pour corriger cette erreur

- Insérez **#End Region** à l'emplacement approprié dans le code après l'instruction **#Region**.

Voir aussi

Référence

#Region, directive

Les instructions '#Region' et '#End Region' ne sont pas valides dans le corps des méthodes

'#Region' and '#End Region' statements are not valid within method bodies

Le bloc **#Region** doit être déclaré au niveau d'une classe, d'un module ou d'un espace de noms. Une région que vous pouvez réduire peut inclure une ou plusieurs procédures, mais elle ne peut pas commencer ou se terminer à l'intérieur d'une procédure.

ID d'erreur : BC32025

Pour corriger cette erreur

1. Assurez-vous qu'une instruction **End Function** ou **End Sub** termine bien la procédure précédente.
2. Assurez-vous que les directives **#Region** et **#End Region** se trouvent dans le même bloc de code.

Voir aussi

Référence

#Region, directive

Le bloc '#If' doit se terminer par un '#End If' correspondant

'#If' block must end with a matching '#End If'

#If est une directive de compilation conditionnelle. Un bloc **#If** n'est pas terminé par une directive **#End If**.

ID d'erreur : BC30012

Pour corriger cette erreur

- Ajoutez une directive **#End If** à la fin du bloc de compilation conditionnelle.

Voir aussi

Référence

#If...Then...#Else, directives

L'instruction '#ExternalSource' doit se terminer par un '#End ExternalSource' correspondant

'#ExternalSource' statement must end with a matching '#End ExternalSource'

La directive **#ExternalSource** fait référence à du code externe, ce qui permet au compilateur de signaler précisément le moment où des exceptions se produisent dans ce code. Un bloc **#ExternalSource** doit commencer par **#ExternalSource** et se terminer par **#End ExternalSource**.

ID d'erreur : BC30579

Pour corriger cette erreur

1. Ajoutez **#End ExternalSource** à l'emplacement approprié dans votre code.
2. Supprimez la **#ExternalSource** initiale si elle n'est pas nécessaire.

Voir aussi

Référence

#ExternalSource, directive

Autres ressources

Compilation conditionnelle (Visual Basic)

Les directives '#ExternalSource' ne peuvent pas être imbriquées

'#ExternalSource' directives cannot be nested

Vous avez tenté de placer une directive **#ExternalSource** dans un autre bloc **#ExternalSource**. La directive **#ExternalSource** fait référence à du code externe, ce qui permet au compilateur de signaler précisément le moment où des exceptions se produisent dans ce code.

Les blocs **#ExternalSource** ne peuvent pas être imbriqués dans d'autres blocs **#ExternalSource**.

ID d'erreur : BC30580

Pour corriger cette erreur

- Déplacez la directive **#ExternalSource** interne en dehors du bloc **#ExternalSource** englobant.

Voir aussi

Référence

#ExternalSource, directive

Autres ressources

Compilation conditionnelle (Visual Basic)

'#End Region' doit être précédé d'un '#Region' correspondant

'#End Region' must be preceded by a matching '#Region'

#Region vous permet de spécifier un bloc de code que vous pouvez développer ou réduire lors de l'utilisation de la fonctionnalité mode Plan de l'éditeur de code de Visual Studio. Le début et la fin des instructions **#Region** doivent figurer dans le même bloc de code.

ID d'erreur : BC30680

Pour corriger cette erreur

- Insérez **#Region** à l'emplacement approprié avant l'instruction **#End Region** correspondante.

Voir aussi

Référence

#Region, directive

'#End ExternalSource' doit être précédé d'un '#ExternalSource' correspondant

'#End ExternalSource' must be preceded by a matching '#ExternalSource'

La directive **#ExternalSource** fait référence à du code externe, ce qui permet au compilateur de signaler précisément le moment où des exceptions se produisent dans ce code. Un bloc **#ExternalSource** doit commencer par **#ExternalSource** et se terminer par **#End ExternalSource**.

ID d'erreur : BC30578

Pour corriger cette erreur

1. Ajoutez **#ExternalSource** à l'emplacement approprié dans votre code.
2. Supprimez **#End ExternalSource** si elle n'est pas nécessaire.

Voir aussi

Référence

#ExternalSource, directive

Autres ressources

Compilation conditionnelle (Visual Basic)

'#ElseIf', '#Else' ou '#End If' doit être précédé d'un '#If' correspondant

'#ElseIf', '#Else', or '#End If' must be preceded by a matching '#If'

#ElseIf, **#Else** et **#End If** sont des directives de compilation conditionnelles. La directive **#ElseIf**, **#Else** ou **#End If** n'est pas précédée d'une directive **#If** correspondante.

ID d'erreur : BC30013

Pour corriger cette erreur

1. Vérifiez que la directive **#If** prévue n'est pas séparée de la clause en question par l'interposition d'un bloc de compilation conditionnelle ou par une directive **#End If** dont l'emplacement est incorrect.

Remarque
Seule une directive **#Else** est autorisée dans chaque bloc **#If** ; par conséquent, deux directives **#Else** successives entraînent cette erreur.

2. Vérifiez que le caractère **#** de début figure bien dans une directive **#If** précédente.
3. Si tout le reste est en ordre, ajoutez une directive **#If** au début du bloc de compilation conditionnelle.

Voir aussi

Référence

#If...Then...#Else, directives

'#ElseIf' doit être précédé d'un '#If' ou '#ElseIf' correspondant

'#ElseIf' must be preceded by a matching '#If' or '#ElseIf'

#ElseIf est une directive de compilation conditionnelle. Une clause **#ElseIf** doit être précédée d'une clause **#If** ou **#ElseIf** correspondante.

ID d'erreur : BC30014

Pour corriger cette erreur

1. Vérifiez qu'une clause **#If** ou **#ElseIf** précédente n'a pas été séparée de cette clause **#ElseIf** par l'interposition d'un bloc de compilation conditionnelle ou par une clause **#End If** dont l'emplacement est incorrect.
2. Si **#ElseIf** est précédée d'une directive **#Else**, supprimez **#Else** ou remplacez-la par **#ElseIf**.
3. Si tout le reste est en ordre, ajoutez une directive **#If** au début du bloc de compilation conditionnelle.

Voir aussi

Référence

#If...Then...#Else, directives

'#ElseIf' ne peut pas suivre '#Else' dans un bloc '#If'

'#ElseIf' cannot follow '#Else' as part of an '#If' block

Une directive de compilation conditionnelle **#ElseIf** suit une directive **#Else**. **#Else** doit être la dernière directive du bloc conditionnel avant la directive **#End If**.

ID d'erreur : BC32030

Pour corriger cette erreur

1. Vérifiez si vous ne devriez pas remplacer la directive **#Else** précédente par **#ElseIf**.
2. Vérifiez qu'un bloc **#If** précédent est correctement terminé et qu'un nouveau bloc **#If** commence.
3. Si tout le reste est correct, déplacez cette directive **#ElseIf** et son bloc d'instructions correspondant de sorte qu'il précède le bloc **#Else**.

Voir aussi

Référence

#If...Then...#Else, directives

'#Else' doit être précédé d'un '#If' ou '#ElseIf' correspondant

'#Else' must be preceded by a matching '#If' or '#ElseIf'

#Else est une directive de compilation conditionnelle. Une directive **#Else** n'est pas précédée d'une directive **#If** ou **#ElseIf** correspondante.

ID d'erreur : BC30028

Pour corriger cette erreur

1. Vérifiez qu'une directive **#If** ou **#ElseIf** précédente n'est pas séparée de cette directive **#Else** par l'interposition d'un bloc de compilation conditionnelle ou par une directive **#End If** dont l'emplacement est incorrect.
2. Vérifiez que la directive **#Else** est précédée d'une autre directive **#Else**. Si tel est le cas, supprimez la directive **#Else** ou remplacez-la par une directive **#ElseIf**.
3. Si tout le reste est en ordre, ajoutez une directive **#If** au début du bloc de compilation conditionnelle.

Voir aussi

Référence

#If...Then...#Else, directives

'!' requiert un opérande gauche ayant un type paramètre de type, classe ou interface, mais cet opérande a le type '<type>'

'!' requires its left operand to have a type parameter, class or interface type, but this operand has the type '<type>'

L'accès au membre dictionnaire, qui utilise un point d'exclamation (**!**) à la place d'un point (**.**), est valide uniquement sur une classe ou une interface.

ID d'erreur : BC30103

Pour corriger cette erreur

- Remplacez l'expression à gauche du point d'exclamation par une expression qui correspond à un type classe ou interface défini.

Voir aussi

Concepts

Caractères spéciaux dans le code

'-' attendu

'-' expected

La syntaxe de cette ligne est incorrecte.

ID d'erreur : BC30601

Pour corriger cette erreur

- Vérifiez la syntaxe du code à l'origine de cette erreur.

Voir aussi

Autres ressources

Structure de programme et conventions de codage

Accès d'un membre partagé via une instance ; l'expression qualifiante ne sera pas évaluée

Access of shared member through an instance; qualifying expression will not be evaluated

Une variable d'instance d'une classe ou d'une structure est utilisée pour accéder à une variable, une propriété, une procédure ou un événement **Shared** défini dans cette classe ou dans cette structure. Cet avertissement peut également se produire si une variable d'instance est utilisée pour accéder à un membre implicitement partagé d'une classe ou d'une structure, par exemple une constante, une énumération, ou encore une classe ou une structure imbriquée.

L'objectif du partage d'un membre est de créer une seule copie de ce membre et de la rendre disponible à chaque instance de la classe ou de la structure dans laquelle elle est déclarée. Il est conforme à l'objectif consistant à accéder à un membre **Shared** par le biais du nom de sa classe ou de sa structure, plutôt que par le biais d'une variable qui contient une instance de cette classe ou de cette structure.

L'accès à un membre **Shared** par le biais d'une variable d'instance peut rendre votre code plus difficile à comprendre en occultant le fait que le membre soit **Shared**. En outre, si un tel accès fait partie d'une expression qui exécute d'autres actions, par exemple une procédure **Function** qui retourne une instance du membre partagé, Visual Basic ignore l'expression et toutes les autres actions qui, autrement, seraient exécutées.

Pour obtenir des informations supplémentaires et un exemple, consultez Shared (Visual Basic).

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42025

Pour corriger cette erreur

- Utilisez le nom de la classe ou de la structure qui définit le membre **Shared** pour y accéder.
- Soyez vigilant par rapport aux effets de portée lorsque deux éléments de programmation portent le même nom. L'exemple suivant déclare une variable `testClass` portant le même nom que la classe du même type.

```
Public Class testClass
    Public Shared Sub sayHello()
        MsgBox("Hello")
    End Sub
End Class
Module testModule
    Public Sub Main()
        Dim testClass As testClass
        testClass.sayHello()
    End Sub
End Module
```

Le compilateur interprète l'appel à `sayHello()` comme un accès via la variable `testClass`, parce que la variable a une portée plus restreinte que la classe du même nom.

Voir aussi

Référence

Shared (Visual Basic)

Concepts

Portée dans Visual Basic

Le modificateur d'accès ne peut être appliqué qu'à 'Get' ou 'Set', mais pas aux deux.

Access modifier can only be applied to either 'Get' or Set', but not both

Une déclaration de propriété spécifie les niveaux d'accès de l'instruction Property, l'instruction Get, et l'instruction Set (Visual Basic).

Vous pouvez toujours spécifier un niveau d'accès pour la propriété. Vous pouvez également spécifier un niveau d'accès différent pour au plus l'une de ses procédures de propriété (**Get** ou **Set**), pourvu qu'elle soit plus restrictive que le niveau d'accès de la propriété. Vous ne pouvez pas spécifier des niveaux d'accès pour les deux procédures de propriété.

ID d'erreur : BC31101

Pour corriger cette erreur

- Supprimez le modificateur d'accès de l'instruction **Get** ou de l'instruction **Set**.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Le modificateur d'accès '<ModificateurAccès>' n'est pas valide

Access modifier '<accessmodifier>' is not valid

Instruction Get ou instruction Set (Visual Basic) spécifie un niveau d'accès qui est moins restrictif que celui spécifié pour la propriété conteneur.

Vous pouvez toujours spécifier un niveau d'accès pour la propriété. De plus, vous pouvez spécifier un niveau d'accès différent pour l'une de ses procédures de propriété (**Get** ou **Set**), à condition qu'il soit plus restrictif que le niveau d'accès de la propriété. Par exemple, si la propriété a la valeur **Friend**, vous pouvez spécifier **Private** pour une procédure de propriété, mais pas **Public**. Vous ne pouvez pas spécifier de niveaux d'accès pour les deux procédures de propriété.

ID d'erreur : BC31100

Pour corriger cette erreur

- Définissez un niveau d'accès pour la procédure de propriété qui soit plus restrictif que celui de la propriété, ou supprimez l'intégralité du modificateur d'accès.
- Déclarez le niveau d'accès moins restrictif dans l'instruction Property, et déclarez le niveau d'accès plus restrictif dans l'une des procédures de propriété.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

L'alias '<nomespacedenoms>' est déjà déclaré

Alias '<namespacename>' is already declared

Vous avez tenté d'utiliser **Imports** pour importer un espace de noms déjà déclaré.

ID d'erreur : BC30572

Pour corriger cette erreur

- Vérifiez l'espace de noms que vous essayez d'importer.

Voir aussi

Référence

Imports, instruction

Alias

L'opérande 'AddressOf' doit être le nom d'une méthode (sans parenthèses)

'AddressOf' operand must be the name of a method (without parentheses)

L'opérateur **AddressOf** crée une instance déléguée de procédure qui fait référence à une procédure spécifique. La syntaxe est la suivante.

AddressOf *procedurename*

Vous avez inséré des parenthèses autour de l'argument qui suit **AddressOf**, alors qu'aucune parenthèse n'est requise.

ID d'erreur : BC30577

Pour corriger cette erreur

1. Supprimez les parenthèses autour de l'argument qui suit **AddressOf**.
2. Vérifiez que l'argument est un nom de méthode.

Voir aussi

Référence

AddressOf, opérateur

Concepts

Délégués et opérateur AddressOf

Les expressions 'AddressOf' ne sont pas valides dans les fenêtres de débogage

'AddressOf' expressions are not valid in debug windows

Les instructions **AddressOf** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30731

Pour corriger cette erreur

- Supprimez les instructions **AddressOf** du code de débogage.

Voir aussi

Référence

AddressOf, opérateur

Autres ressources

Débogage dans Visual Studio

L'expression 'AddressOf' ne peut pas être convertie en '<nomtype>', car le type '<nomtype>' est déclaré 'MustInherit' et ne peut pas être créé

'AddressOf' expression cannot be converted to '<typename>' because type '<typename>' is declared 'MustInherit' and cannot be created

Une instruction tente de convertir une expression **AddressOf** en un type qui peut être uniquement une classe de base et qui ne peut pas être utilisé pour créer une instance.

L'opérateur **AddressOf** crée une instance déléguée de procédure qui fait référence à une procédure spécifique.

ID d'erreur : BC30939

Pour corriger cette erreur

- Assignez l'expression **AddressOf** à un type délégué spécifique.

Voir aussi

Référence
AddressOf, opérateur
Concepts
Délégués et opérateur AddressOf
Autres ressources
Délégués en Visual Basic

L'expression 'AddressOf' ne peut pas être convertie en '<nomtype>' car '<nomtype>' n'est pas un type délégué

'AddressOf' expression cannot be converted to '<typename>' because '<typename>' is not a delegate type

Une instruction tente de convertir une expression **AddressOf** en un type qui n'est pas un type délégué.

L'opérateur **AddressOf** crée une instance déléguée de procédure qui fait référence à une procédure spécifique. **AddressOf** peut être utilisé comme opérande d'un constructeur délégué ou bien dans un contexte où le type du délégué peut être déterminé par le compilateur.

ID d'erreur : BC30581

Pour corriger cette erreur

- Remplacez le type cible en type délégué.

Voir aussi

Référence

AddressOf, opérateur

Concepts

Délégués et opérateur AddressOf

Les paramètres de méthode 'AddHandler', 'RemoveHandler' et 'RaiseEvent' ne peuvent pas être déclarés '<modificateur>'

'AddHandler', 'RemoveHandler' and 'RaiseEvent' method parameters cannot be declared '<modifier>'

Les paramètres des méthodes **AddHandler**, **RemoveHandler** et **RaiseEvent** ne peuvent pas être déclarés avec les modificateurs **Optional** ou **ParamArray**.

Les modificateurs **Optional** ou **ParamArray** ne sont autorisés que pour des paramètres dans les déclarations **Declare**, **Function**, **Property** et **Sub**.

ID d'erreur : BC31138

Pour corriger cette erreur

- Supprimez le mot clé **Optional** ou **ParamArray** de la liste de paramètres.

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler
RaiseEvent
Optional (Visual Basic)
ParamArray

Autres ressources

Événements en Visual Basic

L'opérande d'événement d'instruction 'AddHandler' ou 'RemoveHandler' doit être une expression qualifiée par un point ou un nom simple

'AddHandler' or 'RemoveHandler' statement event operand must be a dot-qualified expression or a simple name

L'élément spécifié pour l'argument d'événement pour **AddHandler** ou **RemoveHandler** n'est pas reconnu en tant qu'événement.

ID d'erreur : BC30677

Pour corriger cette erreur

- Spécifiez le nom de l'objet qui déclenche l'événement suivi par un point (`.`) et le nom de l'événement, comme dans l'exemple suivant.

  VB

```
' Assume that the class Wobject has an event named ThisEvent.
Dim wObj As New Wobject
' Assume that this class has as method named ThisEventHandler.
AddHandler wObj.ThisEvent, AddressOf Me.ThisEventHandler
```

Voir aussi

Référence

AddHandler, instruction

RemoveHandler, instruction

Concepts

AddHandler et RemoveHandler

Autres ressources

Événements en Visual Basic

'AddHandler' est déjà déclaré

'AddHandler' is already declared

Plusieurs déclarations **AddHandler** apparaissent dans une déclaration event personnalisée. Une déclaration **AddHandler** déclare la procédure utilisée pour ajouter un gestionnaire d'événements.

ID d'erreur : BC31127

Pour corriger cette erreur

- Supprimez l'instruction **AddHandler** redondante.

Voir aussi

Référence

AddHandler, instruction
Event, instruction

La définition 'AddHandler' est manquante pour l'événement '<nomévénement>'

'AddHandler' definition missing for event '<eventname>'

Si un événement est déclaré en tant que **Custom**, il doit fournir une procédure pour ajouter un gestionnaire d'événements.

ID d'erreur : BC31130

Pour corriger cette erreur

1. Incluez une déclaration **AddHandler** entre l'instruction **Custom Event** et l'instruction **End Event**.
2. Vérifiez que les autres procédures à l'intérieur de la déclaration event se terminent correctement.

Voir aussi

Référence

AddHandler, instruction
Event, instruction

La déclaration 'AddHandler' doit se terminer par un 'End AddHandler' correspondant

'AddHandler' declaration must end with a matching 'End AddHandler'

Une déclaration **AddHandler** doit se terminer par une instruction **End AddHandler**

ID d'erreur : BC31115

Pour corriger cette erreur

- Vérifiez que la déclaration **AddHandler** se termine par une instruction **End AddHandler**

Voir aussi

Référence

AddHandler, instruction
Event, instruction

Les méthodes 'AddHandler' et 'RemoveHandler' doivent avoir exactement un paramètre

'AddHandler' and 'RemoveHandler' methods must have exactly one parameter

Une déclaration event personnalisée doit posséder les déclarations **AddHandler** ou **RemoveHandler**, chacune d'entre elles prenant un paramètre unique du type délégué spécifié par la clause **As** de l'événement personnalisé.

ID d'erreur : BC31133

Pour corriger cette erreur

- Supprimez les paramètres supplémentaires de la liste de paramètres et modifiez le type de paramètre de sorte qu'il soit le même que le type délégué spécifié par la clause **As** de l'événement personnalisé.

Exemple

Cet exemple illustre un événement personnalisé avec les types de paramètres corrects pour les déclarations **AddHandler** et **RemoveHandler**.

VB

```
Custom Event Test As System.EventHandler
    AddHandler(ByVal value As System.EventHandler)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As System.EventHandler)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler

Autres ressources

Événements en Visual Basic

Les paramètres de méthode 'AddHandler' et 'RemoveHandler' doivent avoir le même type délégué que l'événement conteneur

'AddHandler' and 'RemoveHandler' method parameters must have the same delegate type as the containing event

Une déclaration **Custom Event** doit posséder les déclarations **AddHandler** ou **RemoveHandler**, chacune d'entre elles prenant un paramètre unique du type délégué spécifié par la clause **As** de l'événement personnalisé.

ID d'erreur : BC31136

Pour corriger cette erreur

- Modifiez le type du paramètre de sorte qu'il soit le même que le type délégué spécifié par la clause **As** de l'événement personnalisé.

Exemple

Cet exemple illustre un événement personnalisé avec les types de paramètres corrects pour les déclarations **AddHandler** et **RemoveHandler**.

VB

```
Custom Event Test As System.EventHandler
    AddHandler(ByVal value As System.EventHandler)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As System.EventHandler)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler

Autres ressources

Événements en Visual Basic

Les paramètres de méthode 'AddHandler' et 'RemoveHandler' ne peuvent pas être déclarés 'ByRef'

'AddHandler' and 'RemoveHandler' method parameters cannot be declared 'ByRef'

Les paramètres des méthodes **AddHandler** et **RemoveHandler** ne peuvent pas être déclarés avec le modificateur **ByRef**.

ID d'erreur : BC31134

Pour corriger cette erreur

- Supprimez le mot clé **ByRef** du paramètre.

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler
ByRef

Autres ressources

Événements en Visual Basic

Si l'un des paramètres est explicitement typé, tous doivent l'être

All parameters must be explicitly typed if any are

Vous avez mélangé divers types de déclarations de type dans une fonction.

ID d'erreur : BC30529

Pour corriger cette erreur

- Vérifiez que tous les paramètres sont typés explicitement.

Voir aussi

Référence

Type

Tous les projets d'une solution Visual Basic doivent cibler la même plate-forme, mais le projet que vous essayez d'ajouter cible une plate-forme différente de celle spécifiée par votre solution

All projects in a Visual Basic solution must target the same platform, but the project you are trying to add targets a platform other than the one specified by your solution

Tous les projets d'une solution Visual Basic doivent cibler la même plate-forme, mais le projet que vous ajoutez cible une plate-forme différente de celle spécifiée par votre solution.

ID d'erreur : BC30765

Pour corriger cette erreur

- Ajoutez uniquement des projets qui ciblent la même plate-forme que celle spécifiée par votre solution.

Voir aussi

Tâches

Comment : créer des applications smart device à l'aide de Visual Basic ou Visual C#

L'argument qui correspond au paramètre'<nomparamètre>' passe de '<type1>' à '<type2>'

Argument matching parameter '<parametername>' narrows from '<type1>' to '<type2>'

Vous avez effectué un appel vers une méthode surchargée, mais le compilateur ne peut pas trouver de méthode pouvant être appelée sans conversion restrictive. Dans une conversion restrictive, la valeur est convertie dans un type de données susceptible de ne pas pouvoir accueillir avec précision certaines valeurs possibles des données.

ID d'erreur : BC30520

Pour corriger cette erreur

- Spécifiez **Option Strict Off**.

Voir aussi

Référence

Option Strict, instruction

Concepts

Propriétés et méthodes surchargées

Conversions étendues et restrictives

Argument non spécifié pour le paramètre '<nomparamètre>' de '<nomméthode>'

Argument not specified for parameter '<parametername>' of '<methodname>'

Vous n'avez pas fourni d'argument pour un paramètre requis.

ID d'erreur : BC30455

Pour corriger cette erreur

- Fournissez un argument pour le paramètre spécifié.

Voir aussi

Tâches

Procédures de dépannage

Concepts

Méthodes de classe

Les déclarations de tableau ne peuvent pas spécifier de limites inférieures

Array declarations cannot specify lower bounds

Les tableaux ont toujours une limite inférieure égale à zéro. Vous pouvez spécifier zéro comme limite inférieure pour rendre votre code plus lisible. Toutefois, vous ne pouvez pas spécifier d'autre valeur pour la limite inférieure.

ID d'erreur : BC30805

Pour corriger cette erreur

- Attribuez aux tableaux à une seule dimension une limite inférieure de un au nombre total d'éléments. `Dim y(6)`, par exemple, a la même taille (7 éléments) que `Dim x(3 To 9)`. Vous pouvez également spécifier `Dim y(0 To 6)`.
- Utilisez des offsets pour simuler les limites inférieures différentes de zéro. L'exemple suivant simule un tableau dont les dimensions sont comprises entre 3 et 9.

```
Const offset As Integer = 3
Dim arrayIndex As Integer
' arrayIndex can vary between 3 and 9.
Dim y(0 To 6)
' The preceding statement allocates the same number of elements
' as Dim y(3 To 9).
y(arrayIndex - offset) = value
' The preceding statement converts arrayIndex to the
' corresponding index of y.
```

Voir aussi

Tâches

Comment : spécifier une limite inférieure égale à zéro dans un tableau

Concepts

Dimensions du tableau dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Un tableau déclaré en tant que variable de contrôle de boucle for ne peut pas être déclaré avec une taille initiale

Array declared as for loop control variable cannot be declared with an initial size

Une boucle **For Each** utilise un tableau comme sa variable d'itération *element*, mais initialise ce tableau.

Les instructions suivantes expliquent comment cette erreur peut être générée :

```
Dim arrayList As New List(Of Integer())
For Each listElement() As Integer In arrayList
For Each listElement(1) As Integer In arrayList
```

La première instruction **For Each** désigne la méthode d'accès correcte aux éléments de `arrayList`. La deuxième instruction **For Each** génère cette erreur.

ID d'erreur : BC32039

Pour corriger cette erreur

- Supprimez l'initialisation de la déclaration de la variable d'itération *element*.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Collections en Visual Basic

Les tailles de tableau ne peuvent pas figurer dans les spécificateurs de type

Array bounds cannot appear in type specifiers

Les tailles de tableau ne peuvent pas être déclarées comme partie d'un spécificateur de type de données.

ID d'erreur : BC30638

Pour corriger cette erreur

- Spécifiez la taille du tableau immédiatement après le nom de variable au lieu de la placer après le type, comme le montre l'exemple suivant.

```
Dim Array(8) As Integer
```

- Définissez un tableau et initialisez-le avec le nombre d'éléments souhaité, comme le montre l'exemple suivant.

```
Dim Array2() As Integer = New Integer(8) {}
```

Voir aussi

Tâches

Comment : déclarer une variable tableau

Concepts

Déclaration de la taille des tableaux pour les utilisateurs de Visual Basic 6.0

Autres ressources

Tableaux dans Visual Basic

Les arguments ne peuvent pas être passés à un 'New' utilisé pour un paramètre de type

Arguments cannot be passed to a 'New' used on a type parameter

Une instruction de déclaration ou d'assignation appelle un type générique et fournit des arguments de constructeur à un paramètre de type qui contient la contrainte New (Visual Basic).

Une liste de contraintes sur un paramètre de type peut spécifier que l'argument de type passé à ce paramètre de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder. Un paramètre de type ne peut pas exiger un constructeur qui accepte des paramètres, et un paramètre de type contenant la contrainte **New** ne peut pas accepter ce constructeur.

ID d'erreur : BC32085

Pour corriger cette erreur

- Supprimez la liste d'arguments qui suit l'argument de type dans l'instruction qui appelle le type générique. Vous ne pouvez pas passer des arguments de constructeur au paramètre de type correspondant.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

Les dimensions d'un tableau ne peuvent pas être de taille négative

Array dimensions cannot have a negative size

Une ou plusieurs des limites de tableau spécifiées sont représentées par un nombre négatif. Vous pouvez spécifier un indice négatif uniquement lorsque vous utilisez une limite supérieure égale à -1 pour déclarer un tableau vide. Ce type de tableau n'a aucun élément, mais il n'a pas la valeur **Nothing**.

ID d'erreur : BC30611

Pour corriger cette erreur

- Remplacez la limite de tableau négative par un nombre positif.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Le tableau dépasse le nombre de 32 dimensions autorisées

Array exceeds the limit of 32 dimensions

Les tableaux ne peuvent pas comporter plus de 32 dimensions.

ID d'erreur : BC30052

Pour corriger cette erreur

- Réduisez le nombre de dimensions dans la déclaration du tableau.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Un initialiseur de tableau ne peut pas être spécifié pour une dimension non constante ; utilisez l'initialiseur vide '{}'

Array initializer cannot be specified for a non constant dimension; use the empty initializer '{}'

Un tableau initialise une dimension inconnue au moment de la compilation.

Le code suivant génère cette erreur.

```
Dim j As Integer
Dim intArray As Integer = New Integer(1, j) {{0, 100}, {1,101}}
```

Le code suivant permet d'éviter cette erreur.

```
Dim intArray As Integer = New Integer(1, j) {}
For i As Integer = 0 To j
    intArray(0, i) = i
    intArray(1, i) = 100 + i
Next i
```

ID d'erreur : BC30949

Pour corriger cette erreur

- Si possible, spécifiez une dimension constante dans la déclaration de tableau.
- Si vous ne pouvez pas spécifier de dimension constante, vous devez initialiser le tableau à l'aide d'une boucle lorsque la dimension non constante devient connue.

Voir aussi

Tâches

Comment : initialiser une variable tableau

Comment : initialiser un tableau multidimensionnel

Référence

For Each...Next, instruction (Visual Basic)

Concepts

Vue d'ensemble des tableaux dans Visual Basic

L'initialiseur de tableau utilise <nombre> élément(s) de trop

Array initializer has <number> too many elements

Votre initialiseur de tableau contient trop d'éléments.

ID d'erreur : BC30568

Pour corriger cette erreur

- Utilisez **ReDim** pour modifier la taille du tableau.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic

L'initialiseur de tableau n'utilise pas suffisamment de dimensions

Array initializer has too few dimensions

Vous n'avez pas fourni suffisamment de dimensions dans l'initialiseur de votre tableau.

ID d'erreur : BC30565

Pour corriger cette erreur

- Vérifiez votre initialiseur de tableau pour déterminer le nombre de dimensions nécessaire.
- Spécifiez le ou les arguments manquants.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic

L'initialiseur de tableau utilise trop de dimensions

Array initializer has too many dimensions

Vous avez fourni trop de dimensions dans l'initialiseur de votre tableau.

ID d'erreur : BC30566

Pour corriger cette erreur

- Vérifiez votre initialiseur de tableau pour déterminer le nombre de dimensions nécessaire.
- Supprimez le ou les arguments supplémentaires.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic
Tableaux en escalier dans Visual Basic

Il manque <nombre> éléments à l'initialiseur de tableau

Array initializer is missing <number> elements

Vous n'avez pas fourni suffisamment d'éléments dans l'initialiseur de votre tableau.

ID d'erreur : BC30567

Pour corriger cette erreur

- Spécifiez le ou les arguments manquants.
- Utilisez **ReDim** pour modifier la taille du tableau.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux en escalier dans Visual Basic

Les initialiseurs de tableaux sont valides uniquement pour les tableaux, mais le type de '<nomvariable>' est '<nomtype>'

Array initializers are valid only for arrays, but the type of '<variablename>' is '<typename>'

Une tentative d'initialisation d'une variable autre qu'une variable tableau à l'aide d'une liste de valeurs a été effectuée.

ID d'erreur : BC30679

Pour corriger cette erreur

- Déclarez et initialisez la variable en tant que tableau. Par exemple :

  ```
  Dim intarray As Integer() = {1, 5, 9}
  ```

- Initialisez la variable en tant que valeur unique. Par exemple :

  ```
  Dim intvalue As Integer = 1
  ```

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic

Autres ressources

Tableaux dans Visual Basic

Les limites inférieures du tableau ne peuvent être que '0'

Array lower bounds can be only '0'

Une instruction de déclaration ou une clause **New** spécifie un tableau avec une limite inférieure autre que 0.

Lorsque vous créez ou initialisez une variable tableau, vous pouvez éventuellement spécifier la limite supérieure de chaque dimension. Dans ce cas, la longueur de cette dimension devient la limite supérieure plus un, parce que la limite inférieure est toujours nulle. De même, vous pouvez éventuellement définir la limite inférieure, mais vous devez spécifier 0. L'avantage de cette approche est que votre code est plus facile à lire.

ID d'erreur : BC32059

Pour corriger cette erreur

- Remplacez la spécification de limite inférieure par 0 ou supprimez-la complètement.

Voir aussi

Tâches

Comment : spécifier une limite inférieure égale à zéro dans un tableau

Autres ressources

Tableaux dans Visual Basic

Impossible de spécifier des modificateurs de tableau sur la variable et son type

Array modifiers cannot be specified on both a variable and its type

Les modificateurs de tableau sont présents à la fois sur la variable et son type, ce qui indique un tableau de tableaux.

ID d'erreur : BC31087

Pour corriger cette erreur

- Supprimez le modificateur de tableau du spécificateur de type.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Expression d'indice de tableau manquante

Array subscript expression missing

Une initialisation de tableau omet un ou plusieurs des indices qui définissent les tailles de tableau. Par exemple, l'instruction peut contenir l'expression `myArray (5,5,,10)` qui omet le troisième indice.

ID d'erreur : BC30306

Pour corriger cette erreur

- Spécifiez l'indice manquant.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Les tableaux ne peuvent pas être déclarés avec 'New'

Arrays cannot be declared with 'New'

Le mot clé **New** peut apparaître uniquement dans la partie initialisation d'une déclaration de tableau. En d'autres termes, **New** doit être placé à droite du signe égal (**=**), de manière à ce qu'il puisse créer un nouveau type tableau à assigner à la variable tableau.

Le raccourci pour l'initialisation de classe n'est pas disponible pour les tableaux. Les deux lignes de code suivantes sont valides et équivalentes parce qu'elles initialisent un objet d'une classe.

```
Dim formA as Form = New Form
Dim formA as New Form
```

Toutefois, une initialisation de tableau ne peut pas utiliser le même raccourci qu'une initialisation de classe.

Notez que la clause **New** pour un tableau doit contenir deux parenthèses, **()**, et des accolades, **{}**. Les parenthèses spécifient que le nouveau type est un tableau, et les accolades fournissent les valeurs d'initialisation. Le compilateur exige la présence d'accolades même si celles-ci sont vides, c'est-à-dire si vous n'initialisez aucune des valeurs du tableau.

ID d'erreur : BC30053

Pour corriger cette erreur

- Remplacez une instruction, telle que `Dim myDates() As New Date` par une instruction, telle que `Dim myDates() As Date = New Date() {}`.

Voir aussi

Tâches

Comment : initialiser une variable tableau

Autres ressources

Tableaux dans Visual Basic

Les tableaux déclarés en tant que membres de structures ne peuvent pas être déclarés avec une taille initiale

Arrays declared as structure members cannot be declared with an initial size

Un tableau dans une structure est déclaré avec une taille initiale. Vous ne pouvez pas initialiser d'élément de structure. À ce titre, vous ne pouvez pas déclarer une taille de tableau car il s'agit d'une forme d'initialisation.

ID d'erreur : BC31043

Pour corriger cette erreur

1. Définissez le tableau dans votre structure en tant que dynamique (sans taille initiale).
2. Si vous avez besoin d'une certaine taille de tableau, vous pouvez redimensionner un tableau dynamique avec ReDim, instruction (Visual Basic) pendant l'exécution de votre code. L'exemple suivant illustre ce comportement :

```
Structure demoStruct
    Public demoArray() As Integer
End Structure
Sub useStruct()
    Dim struct As demoStruct
    ReDim struct.demoArray(9)
    Struct.demoArray(2) = 777
End Sub
```

Voir aussi

Tâches

Comment : déclarer une structure

Autres ressources

Tableaux dans Visual Basic

Les tableaux de type 'System.Void' ne sont pas autorisés dans cette expression

Arrays of type 'System.Void' are not allowed in this expression

Une expression dans une instruction d'assignation ou une déclaration spécifie un tableau de type Void.

La structure **Void** est un type spécialisé utilisé en interne par le .NET Framework et en particulier par Visual C# et Visual C++. Elle représente un type valeur de retour pour une méthode qui ne retourne pas de valeur. Visual Basic utilise une procédure **Sub** lorsqu'aucune valeur n'est retournée et une procédure **Function** lorsque une valeur est retournée.

Les tableaux de type **Void** ne sont pas explicites et ne sont pas autorisés dans n'importe quel contexte.

ID d'erreur : BC31428

Pour corriger cette erreur

1. Supprimez les parenthèses qui désignent un tableau.
2. À moins que vous ayez une raison particulière de comparer un type d'exécution à **Void**, supprimez entièrement la référence à ce type.

Voir aussi

Référence

Void

Les tableaux utilisés en tant qu'arguments d'attribut sont requis pour spécifier explicitement des valeurs pour tous les éléments

Arrays used as attribute arguments are required to explicitly specify values for all elements

Un tableau utilisé en tant qu'argument d'attribut possède un élément avec une valeur non spécifiée.

ID d'erreur : BC31110

Pour corriger cette erreur

- Spécifiez toutes les valeurs des éléments.

Voir aussi

Tâches

Dépannage des tableaux

Autres ressources

Tableaux dans Visual Basic

'As Any' n'est pas pris en charge dans les instructions 'Declare'

'As Any' is not supported in 'Declare' statements

Le type de données **Any** était utilisé avec des instructions **Declare** dans les versions antérieures de Visual Basic, afin de permettre l'utilisation d'arguments pouvant contenir tout type de données. Visual Basic prend cependant en charge les surcharges ; le type de données **Any** devient donc obsolète.

ID d'erreur : BC30828

Pour corriger cette erreur

1. Déclarez les paramètres du type spécifique que vous souhaitez utiliser, comme dans l'exemple suivant.

   VB

   ```
   Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" ( _
       ByVal lpBuffer As String, _
       ByRef nSize As Integer) _
       As Integer
   ```

2. Utilisez l'attribut MarshalAsAttribute pour spécifier **As Any** lorsque **Void*** est attendu par la procédure appelée.

   VB

   ```
   Declare Sub SetData Lib "..\LIB\UnmgdLib.dll" ( _
       ByVal x As Short, _
       <System.Runtime.InteropServices.MarshalAsAttribute( _
           System.Runtime.InteropServices.UnmanagedType.AsAny)> _
           ByVal o As Object)
   ```

Voir aussi

Tâches

Procédure pas à pas : appel des API Windows

Référence

Declare, instruction

MarshalAsAttribute

Concepts

Création de prototypes dans du code managé

'As' attendu

'As' expected

Une déclaration apparaît sans fournir de clause **As** pour un ou plusieurs éléments déclarés.

ID d'erreur : BC30197

Pour corriger cette erreur

- Fournissez la clause **As** requise ou définissez **Option Strict Off**.

Voir aussi

Référence

As (Visual Basic)

Option Strict, instruction

'As', virgule ou ')' attendu

'As', comma or ')' expected

Le paramètre de type qui a été fourni ne contient pas de **As**, de virgule (`,`) ou de parenthèse fermante (`)`).

ID d'erreur : BC32100

Pour corriger cette erreur

- Entrez l'élément manquant : **As**, virgule ou parenthèse fermante.

Voir aussi

Tâches

Dépannage des types de données

Concepts

Types d'erreurs

Impossible de créer l'assembly '<nomassembly>', car son chemin d'accès excède 259 caractères

Assembly '<assemblyname>' cannot be created because its path is longer than 259 characters

Le chemin d'accès de cet assembly excède 259 caractères, longueur maximale autorisée.

ID d'erreur : BC31518

Pour corriger cette erreur

- Créez un chemin d'accès plus court en utilisant des noms plus courts ou en sélectionnant une hiérarchie de fichiers plus superficielle.

Voir aussi

Concepts

Assemblys

L'assembly '<CheminFichier1>' référence l'assembly '<IdentitéAssembly>', ce qui crée une ambiguïté entre '<CheminFichier2>' (référencé par le projet '<NomProjet1>') et '<CheminFichier3>' (référencé par le projet '<NomProjet2>')

Message d'erreur
L'assembly '<CheminFichier1>' référence l'assembly '<IdentitéAssembly>', ce qui crée une ambiguïté entre '<CheminFichier2>' (référencé par le projet '<NomProjet1>') et '<CheminFichier3>' (référencé par le projet '<NomProjet2>'). '<CheminFichier2>' sera utilisé. Si les deux assemblys sont identiques, remplacez les références au même emplacement.
Assembly '<filepath1>' references assembly '<assemblyidentity>', which is ambiguous between '<filepath2>' (referenced by project '<projectname1>') and '<filepath3>' (referenced by project '<projectname2>'). '<filepath2>' will be used. If both assemblies are identical, change the references to the same location.

Un assembly accède à un type dans un autre assembly auquel il a plusieurs références de fichier.

Le compilateur ne peut pas garantir que les fichiers aux différents emplacements contiennent la même version du même assembly. Par conséquent, les références de fichier sont ambiguës et le compilateur doit en sélectionner une.

L'*identité de l'assembly* inclut son nom, sa version, sa clé publique, si elle existe, et sa culture. Ces informations identifient l'assembly de manière unique.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42204

Pour corriger cette erreur

1. Déterminez le fichier qui convient le mieux pour l'assembly. Vous pouvez utiliser des critères tels que la version la plus récente, l'accessibilité du fichier et la probabilité de mise à jour selon les besoins.
2. Modifiez toutes les références de fichier à cet assembly afin qu'elles utilisent le même chemin d'accès vers votre fichier sélectionné.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Assemblys
Vue d'ensemble des assemblys
Avantages des assemblys
Références de projet
Autres ressources
Gestion des références

L'assembly '<CheminFichier1>' référence l'assembly '<IdentitéAssembly>', ce qui crée une ambiguïté entre '<CheminFichier2>' et '<CheminFichier3>'

Message d'erreur
L'assembly '<CheminFichier1>' référence l'assembly '<IdentitéAssembly>', ce qui crée une ambiguïté entre '<CheminFichier2>' et '<CheminFichier3>'. '<CheminFichier2>' sera utilisé.
Assembly '<filepath1>' references assembly '<assemblyidentity>', which is ambiguous between '<filepath2>' and '<filepath3>'. '<filepath2>' will be used.

Un assembly accède à un type d'un autre assembly dont il possède plusieurs références de fichier.

Le compilateur ne peut pas garantir que les fichiers présents dans différents emplacements contiennent la même version du même assembly. Par conséquent, les références de fichier sont ambiguës et le compilateur doit en sélectionner une.

L' *identité d'assembly* inclut le nom, la version, la clé publique (le cas échéant) et la culture de l'assembly. Ces informations ne servent qu'à identifier l'assembly.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42205

Pour corriger cette erreur

1. Déterminez le fichier qui représente le meilleur choix pour l'assembly. Vous pouvez utiliser des critères tels que la version la plus récente, l'accessibilité du fichier et la probabilité qu'il soit mis à jour au moment opportun.
2. Modifiez toutes les références de fichier à cet assembly afin qu'elles utilisent le chemin d'accès identique au fichier que vous avez sélectionné.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Assemblys
Vue d'ensemble des assemblys
Avantages des assemblys
Références de projet
Autres ressources
Gestion des références

Attribut d'assembly '<nomattribut>' non valide : <erreur>

Assembly attribute '<attributename>' is not valid: <error>

Le Common Language Runtime n'accepte pas l'attribut sur un assembly.

ID d'erreur : BC30129

Pour corriger cette erreur

1. Supprimez l'attribut de la déclaration ou remplacez-le par un attribut valide pour le contexte.
2. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.

Voir aussi

Tâches

Comment : définir vos propres attributs

Les instructions d'attribut Assembly ou Module doivent précéder toutes les déclarations dans un fichier

Assembly or Module attribute statements must precede any declarations in a file

Les attributs globaux doivent être déclarés au début d'un fichier source, après les instructions **Option** et **Imports**, mais avant toutes les autres instructions.

ID d'erreur : BC30637

Pour corriger cette erreur

- Placez les attributs globaux, tels que `<Module:>` et `<Assembly:>` au début de votre fichier source.

Voir aussi

Concepts

Attributs globaux en Visual Basic

Autres ressources

Attributs en Visual Basic

'Assembly' ou 'Module' attendu

'Assembly' or 'Module' expected

Un attribut global est spécifié sans indication de sa portée (en d'autres termes, s'il s'applique à l'intégralité de l'assembly ou uniquement au module actuel). Les attributs globaux doivent spécifier **Assembly** ou **Module**.

Un attribut global s'affiche automatiquement sur une ligne source, plutôt que d'être appliqué à la déclaration d'un élément de programmation spécifique.

ID d'erreur : BC32015

Pour corriger cette erreur

1. Si vous souhaitez que l'attribut soit global, ajoutez le mot clé **Assembly** ou **Module** au début du bloc d'attributs, suivi de deux-points (:).
2. Dans le cas contraire, supprimez le bloc d'attributs ou déplacez-le dans une déclaration d'élément de programmation.

Voir aussi

Référence

Assembly

Module (Visual Basic)

Concepts

Application des attributs

Attributs globaux en Visual Basic

Au moins un type de paramètre de cet opérateur binaire doit être le type conteneur '<NomType>'

At least one parameter type of this binary operator must be the containing type '<typename>'

Une définition d'un opérateur binaire spécifie les deux paramètres avec des types autres que ceux de la classe ou la structure dans laquelle l'opérateur est défini.

Lorsque vous définissez un opérateur dans une classe ou une structure, au moins l'un des paramètres doit être du type de cette classe ou structure.

ID d'erreur : BC33021

Pour corriger cette erreur

- Remplacez le type d'un ou des deux paramètres par le type de la classe ou la structure dans laquelle l'opérateur est défini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

L'attribut '<nomattribut>' ne peut pas être appliqué plusieurs fois

Attribute '<attributename>' cannot be applied multiple times

L'attribut ne peut être appliqué qu'une seule fois. L'attribut **AttributeUsage** détermine si un attribut peut être appliqué plusieurs fois.

ID d'erreur : BC30663

Pour corriger cette erreur

1. Vérifiez que l'attribut n'est appliqué qu'une seule fois.
2. Si vous utilisez des attributs personnalisés que vous avez développés, modifiez l'attribut **AttributeUsage** de manière à ce qu'il autorise l'utilisation de plusieurs attributs, comme dans l'exemple suivant.

```
<AttributeUsage(AllowMultiple := True)>
```

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs en Visual Basic

Attributs personnalisés en Visual Basic

L'attribut '<nomattribut>' ne peut pas être appliqué à '<nommembre>', car il n'est pas valide dans ce type de déclaration

Attribute '<attributename>' cannot be applied to '<membername>' because the attribute is not valid on this declaration type

L'attribut utilisé n'est pas approprié à l'élément utilisé.

ID d'erreur : BC30662

Pour corriger cette erreur

1. Choisissez un attribut conçu pour l'élément utilisé. Par exemple, si vous utilisez une méthode, choisissez un attribut conçu pour être utilisé avec des méthodes.
2. Si vous utilisez des attributs personnalisés que vous avez développés, changez l'attribut **AttributeUsage** afin qu'il corresponde au type d'élément utilisé.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs en Visual Basic

Attributs personnalisés en Visual Basic

L'attribut '<nomattribut>' ne peut pas être appliqué à une méthode avec des paramètres optionnels

Attribute '<attributename>' cannot be applied to a method with optional parameters

L'attribut peut seulement être utilisé avec des méthodes utilisant elles-mêmes des arguments requis ou aucun argument.

ID d'erreur : BC30645

Pour corriger cette erreur

1. Définissez la méthode sans paramètre optionnel.
2. Utilisez un attribut qui peut être appliqué à des méthodes possédant des paramètres optionnels.
3. Définissez un attribut personnalisé pouvant être utilisé dans ce contexte.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs en Visual Basic

L'attribut '<nomattribut>' ne peut pas être appliqué à un module

Attribute '<attributename>' cannot be applied to a module

Vous avez essayé d'appliquer un attribut à un module dont **AttributeUsageAttribute** ne spécifie pas **AttributeTargets.Module**. Lorsque l'attribut a été déclaré, il n'a pas été défini pour être appliqué à un module.

ID d'erreur : BC30549

Pour corriger cette erreur

- Vérifiez la déclaration attribute et spécifiez **AttributeTargets.Module** ou **AttributeTargets.All**.

Voir aussi

Référence

AttributeUsageAttribute Members
AttributeTargets Enumeration

L'attribut '<nomattribut>' ne peut pas être appliqué à un assembly

Attribute '<attributename>' cannot be applied to an assembly

Vous avez essayé d'appliquer un attribut à un assembly dont **AttributeUsageAttribute** ne spécifie pas **AttributeTargets.Assembly**. Lorsque l'attribut a été déclaré, il n'a pas été défini comme étant applicable à un assembly.

ID d'erreur : BC30548

Pour corriger cette erreur

- Vérifiez la déclaration attribute et spécifiez **AttributeTargets.Assembly** ou **AttributeTargets.All**.

Voir aussi

Référence

AttributeUsageAttribute Members
AttributeTargets Enumeration

L'attribut '<nomattribut>' ne peut pas être spécifié plusieurs fois dans ce projet, même avec des valeurs de paramètre identiques

Attribute '<attributename>' cannot be specified more than once in this project, even with identical parameter values

Un attribut personnalisé est spécifié plusieurs fois dans le même élément de programmation, mais AttributeUsageAttribute est appliqué à l'attribut personnalisé et sa propriété AllowMultiple a la valeur **False**. **AllowMultiple** contrôle si l'attribut est à usages multiples.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41000

Pour corriger cette erreur

- Supprimez la spécification supplémentaire de l'attribut personnalisé ou affectez à **AllowMultiple** la valeur **True** dans **AttributeUsageAttribute** qui est appliqué.

Voir aussi

Référence

AttributeUsageAttribute

AllowMultiple

Concepts

Application des attributs

Attribut '<nomattribut>' non valide : <erreur>

Attribute '<attributename>' is not valid: <error>

Le Common Language Runtime n'accepte pas l'attribut dans le contexte en cours.

ID d'erreur : BC30127

Pour corriger cette erreur

1. Supprimez l'attribut de la déclaration ou remplacez-le par un attribut valide pour le contexte.
2. Examinez le message d'erreur et prenez les mesures qui s'imposent.

Voir aussi

Tâches

Comment : définir vos propres attributs

Impossible d'utiliser cet attribut, car il n'a pas de constructeur public

Attribute cannot be used because it does not have a Public constructor

Le constructeur de l'attribut utilisé est **Private**. Vous ne pouvez pas l'utiliser.

ID d'erreur : BC30758

Pour corriger cette erreur

- Si cette erreur se produit avec un attribut personnalisé que vous avez développé, remplacez le modificateur d'accès du constructeur **Sub New** par **Public**.

Voir aussi

Concepts

Durée de vie d'un objet : création et destruction des objets

Autres ressources

Attributs en Visual Basic

La constante d'attribut '<nomconstante>' ne peut pas être la cible d'une assignation

Attribute constant '<constantname>' cannot be the target of an assignment

Une tentative d'assignation d'une valeur à une constante déclarée dans un attribut a été effectuée.

ID d'erreur : BC31510

Pour corriger cette erreur

- Supprimez l'assignation erronée.
- Si vous utilisez des attributs personnalisés que vous avez développés, redéfinissez le membre en tant que champ dans l'attribut.

Voir aussi

Autres ressources

Attributs en Visual Basic
Constantes et énumérations (Visual Basic)

Le constructeur d'attribut a un paramètre 'ByRef' de type '<NomType>' ; impossible d'utiliser les constructeurs avec des paramètres 'byref' pour appliquer l'attribut

Attribute constructor has a 'ByRef' parameter of type '<typename>'; cannot use constructors with byref parameters to apply the attribute

Un attribut est appliqué à un élément de programmation à l'aide d'un constructeur d'attribut qui accepte un paramètre **ByRef**.

Les attributs sont appliqués au moment de la compilation, et le compilateur a besoin de valeurs concrètes pour passer au constructeur d'attribut. Un paramètre **ByRef** accepte un pointeur vers une valeur qui ne peut pas être évaluée au moment de la compilation.

Vous pouvez définir un constructeur d'attribut qui accepte un paramètre **ByRef**, et vous pouvez l'utiliser pour les besoins d'héritage. Mais lorsque vous appliquez l'attribut, vous devez utiliser un constructeur qui n'accepte pas les paramètres **ByRef**.

ID d'erreur : BC36006

Pour corriger cette erreur

- Appliquez l'attribut à l'aide d'un constructeur qui n'accepte pas les paramètres **ByRef**, ou ne l'appliquez pas.

Voir aussi

Référence

ByRef

Concepts

Vue d'ensemble des attributs dans Visual Basic
Application des attributs
Passage d'arguments par valeur et par référence

Le constructeur d'attribut a un paramètre de type '<type>', qui n'est pas un type intégral, virgule flottante ni Enum, ni encore Object, Char, String, Boolean, System.Type ou un tableau unidimensionnel de ces types

Attribute constructor has a parameter of type '<type>', which is not an integral, floating-point, or Enum type or one of Char, String, Boolean, System.Type or 1-dimensional array of these types

Une définition d'attribut personnalisée inclut un constructeur qui spécifie un type de données non valide pour un paramètre. Les attributs ne peuvent accepter que certains types de données comme paramètres, car seuls ces types peuvent être sérialisés dans les métadonnées de l'assembly.

ID d'erreur : BC30045

Pour corriger cette erreur

- Remplacez le type de données du paramètre par **Byte**, **Short**, **Integer**, **Long**, **Single**, **Double**, **Char**, **String**, **Boolean**, **System.Type** ou par un type énumération.

Voir aussi

Autres ressources

Attributs personnalisés en Visual Basic

Le membre d'attribut '<nommembre>' ne peut pas être la cible d'une assignation, car il n'est pas déclaré 'Public'

Attribute member '<membername>' cannot be the target of an assignment because it is not declared 'Public'

Une tentative d'assignation d'une valeur à un membre privé d'un attribut a été effectuée.

ID d'erreur : BC31511

Pour corriger cette erreur

1. Supprimez l'assignation.
2. Si vous utilisez des attributs personnalisés que vous avez développés, définissez le modificateur d'accès du membre d'attribut sur **Public**.

Voir aussi

Référence

Public (Visual Basic)

Autres ressources

Attributs en Visual Basic

Le spécificateur d'attribut n'est pas une instruction complète

Message d'erreur
Le spécificateur d'attribut n'est pas une instruction complète. Utilisez un signe de continuation de ligne pour appliquer l'attribut à l'instruction suivante.
Attribute specifier is not a complete statement. Use a line continuation to apply the attribute to the following statement.

Un bloc d'attributs apparaît seul dans une ligne de code source. Les attributs doivent être appliqués au début d'une instruction de déclaration et doivent faire partie de cette instruction.

ID d'erreur : BC32035

Pour corriger cette erreur

- Si l'instruction de déclaration se trouve sur la ligne suivante, ajoutez un espace et un trait de soulignement (_) à la suite du bloc d'attributs afin de combiner les lignes de code source.
- Si aucune instruction de déclaration n'est associée au bloc d'attributs, fournissez-en une ou supprimez le bloc d'attributs.
- Si l'attribut doit s'appliquer à tout l'assembly ou au module d'assembly actuel, le bloc d'attributs reste sur une ligne de code source séparée. Faites précéder le nom de l'attribut à l'intérieur des signes « inférieur à » et « supérieur à » (**< >**) de **Assembly:** ou **Module:**, et n'ajoutez pas d'espace ou de trait de soulignement à la suite du bloc d'attributs. Veillez également à ce que ce bloc d'attributs se trouve au début de votre fichier source.

Voir aussi

Tâches
Comment : diviser et combiner des instructions dans le code
Concepts
Application des attributs

L'attribut 'System.Runtime.InteropServices.DefaultCharSetAttribute' n'est pas pris en charge dans cette version

Attribute 'System.Runtime.InteropServices.DefaultCharSetAttribute' is not supported in this version

L'attribut System.Runtime.InteropServices.DefaultCharSetAttribute vous permet de spécifier le jeu de caractères à utiliser dans les chaînes marshalées. Sa valeur prend un membre de l'énumération System.Runtime.InteropServices.CharSet.

La version actuelle de Visual Basic ne prend pas en charge cet attribut. Une prise en charge est possible dans les prochaines versions.

ID d'erreur : BC32510

Pour corriger cette erreur

- Utilisez chaque instruction Declare pour spécifier le jeu de caractères de la procédure externe qu'il déclare. L'exemple suivant illustre ce comportement :

```
Ansi Declare Function GetUserName Lib "advapi32.dll" _
    (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Unicode Declare Sub externalProc Lib "projectlib.dll" _
    (ByVal arg As Double)
```

  Si vous ne spécifiez pas le jeu de caractères dans l'instruction **Declare**, la valeur prise par défaut est ANSI.

Voir aussi

Référence

Declare, instruction
DefaultCharSetAttribute
CharSet

Autres ressources

Attributs en Visual Basic

Les attributs ne peuvent pas être appliqués aux variables locales

Attributes cannot be applied to local variables

Les attributs ne peuvent pas être appliqués aux variables locales.

ID d'erreur : BC30660

Pour corriger cette erreur

- Déclarez cette variable au niveau de portée immédiatement supérieur, tel que le niveau de classe ou de module.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs en Visual Basic

Les attributs ne peuvent pas être des génériques ou des types imbriqués dans des génériques

Attributes cannot be generics or types nested inside generics

Un attribut est déclaré comme un type générique, ou dans un type générique.

Visual Basic et le .NET Framework ne prennent pas en charge actuellement les combinaisons d'attributs et de types génériques. Cela signifie les restrictions suivantes s'appliquent :

- Un attribut ne peut pas être un type générique ou être déclaré dans un type générique.
- Un attribut ne peut pas hériter d'une classe générique, et inversement.
- Lorsque vous appliquez un attribut, vous ne pouvez pas fournir un argument qui fait partie des éléments suivants :
    - un type générique,
    - un type construit à partir d'un type générique,
    - un paramètre de type d'un type conteneur, ou
    - un type construit à partir d'un paramètre de type d'un type conteneur.

ID d'erreur : BC32067

Pour corriger cette erreur

- Si la déclaration attribute contient le mot clé **Of** et une liste de paramètres de type, supprimez-les.
- Si la déclaration attribute apparaît à l'intérieur d'un type générique, déplacez-la à l'extérieur du type générique.

Voir aussi

Référence

Attribute

Concepts

Vue d'ensemble des attributs dans Visual Basic

Types génériques en Visual Basic

Valeur de checksum erronée, pas de chiffre hexadécimal ou de nombre impair de chiffres hexadécimaux

Bad checksum value, non hex digits or odd number of hex digits

Une valeur de checksum contient des chiffres hexadécimaux incorrects ou un nombre impair de chiffres.

Lorsque ASP.NET génère un fichier source Visual Basic (extension .vb), il calcule un checksum et le place dans un fichier source masqué identifié par `#externalchecksum`. Un utilisateur qui génère un fichier .vb peut également exécuter ce processus, mais il est préférable de le limiter à une utilisation interne.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42033

Pour corriger cette erreur

1. Si ASP.NET génère le fichier source Visual Basic, redémarrez la génération du projet.
2. Si cet avertissement persiste après le redémarrage, réinstallez ASP.NET et recommencez la génération.
3. Si l'avertissement persiste toujours ou si vous n'utilisez pas ASP.NET, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Concepts

Vue d'ensemble d'ASP.NET

Autres ressources

Support technique et accessibilité

Format de GUID incorrect

Bad GUID format

Un GUID a été fourni avec un format incorrect.

ID d'erreur : BC42035

Pour corriger cette erreur

- Vérifiez le format du GUID.

Voir aussi

Concepts

Types d'erreurs

La classe de base '<NomClasseDeBase1>' spécifiée pour la classe '<NomClassePartiel>' ne peut pas être différente de la classe de base '<NomClasseDeBase2>' de l'un de ses autres types partiels

Base class '<baseclassname1>' specified for class '<partialclassname>' cannot be different from the base class '<baseclassname2>' of one of its other partial types

Une classe est définie dans deux ou plusieurs déclarations partielles qui contiennent plusieurs instructions Inherits spécifiant plusieurs classes de base.

Lorsque vous divisez la définition d'une classe en plusieurs déclarations partielles, le compilateur traite la classe comme l'union de toutes ses déclarations partielles. Cette règle s'applique non seulement aux membres mais également à l'implémentation, à l'héritage et au niveau d'accès.

Une classe peut implémenter plusieurs interfaces, mais elle ne peut hériter que d'une seule classe de base. Par conséquent, toutes les instructions **Inherits** doivent spécifier la même classe de base.

ID d'erreur : BC30928

Pour corriger cette erreur

- Déterminez la classe qui doit être la classe de base de votre classe partielle, et supprimez de ses déclarations partielles toutes les instructions **Inherits** qui spécifient une classe de base différente.

Voir aussi

Référence

Partial (Visual Basic)

Inherits, instruction

Concepts

Éléments fondamentaux de l'héritage

Les limites ne peuvent être spécifiées que pour le tableau de premier niveau lors de l'initialisation d'un tableau de tableaux

Bounds can be specified only for the top-level array when initializing an array of arrays

Une initialisation de tableau pour un tableau en escalier (tableau de tableaux) définit la longueur initiale de l'un des niveaux inférieurs. Vous pouvez spécifier la longueur du tableau de niveau supérieur uniquement dans l'instruction de déclaration de tableau.

ID d'erreur : BC32014

Pour corriger cette erreur

1. Supprimez la spécification de longueur de tous les tableaux à l'exception du tableau de niveau supérieur.
2. Définissez la longueur initiale des tableaux de niveau inférieur dans les instructions d'assignation suivantes en utilisant le mot clé **New**.

Voir aussi

Tâches

Comment : déclarer une variable tableau

Référence

New (Visual Basic)

Concepts

Tableaux en escalier dans Visual Basic

Crochet fermant manquant (']') pour l'identificateur

Bracketed identifier is missing closing ']'

Dans un nom d'échappement, les crochets doivent apparaître sous forme de paires correspondantes.

ID d'erreur : BC30034

Pour corriger cette erreur

- Placez un crochet fermant à la fin du nom d'échappement.

Voir aussi

Concepts

Noms d'éléments déclarés

Tout branchement en dehors d'un 'Finally' n'est pas valide

Branching out of a 'Finally' is not valid

Le branchement d'une instruction **GoTo** à l'intérieur d'un bloc **Finally** s'effectue à l'extérieur du bloc. Il n'est pas valide d'effectuer le branchement dans ou hors d'un bloc **Catch** ou **Finally**.

ID d'erreur : BC30101

Pour corriger cette erreur

- Supprimez l'instruction **GoTo** et implémentez la logique du programme avec des structures de décision ou de contrôle de boucle.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

GoTo, instruction

Autres ressources

Flux de contrôle en Visual Basic

Impossible de combiner 'ByVal' et 'ByRef'

'ByVal' and 'ByRef' cannot be combined

Les mots clés **ByVal** et **ByRef** s'excluent mutuellement.

ID d'erreur : BC30641

Pour corriger cette erreur

- Spécifiez **ByVal** ou **ByRef**, mais pas les deux.

Voir aussi

Référence

ByVal
ByRef

Impossible de convertir en '<type>'

Cannot convert to '<type>'

Une tentative de conversion non valide a eu lieu.

ID d'erreur : BC30748

Pour corriger cette erreur

- Choisissez un type de conversion compatible.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

Impossible de copier la valeur du paramètre 'ByRef' '<NomParamètre>' dans l'argument correspondant, car le type '<NomType1>' ne peut pas être converti en type '<NomType2>'

Cannot copy the value of 'ByRef' parameter '<parametername>' back to the matching argument because type '<typename1>' cannot be converted to type '<typename2>'

Une procédure est déclarée avec un type de paramètre qui ne peut pas être converti en type d'argument d'appel.

Lorsque vous définissez une classe ou une structure, vous pouvez définir un ou plusieurs opérateurs de conversion pour convertir ce type de classe ou de structure en d'autres types. Vous pouvez également définir des opérateurs de conversion inverse pour convertir ces autres types en votre type de classe ou de structure. Lorsque vous utilisez votre type de classe ou de structure dans un appel de procédure, Visual Basic peut utiliser ces opérateurs de conversion pour convertir le type d'un argument en type de son paramètre correspondant.

Si vous passez l'argument ByRef, Visual Basic copie parfois sa valeur dans une variable locale de la procédure au lieu de passer une référence. Dans ce cas, lors du retour de la procédure, Visual Basic doit copier la valeur de la variable locale dans l'argument du code appelant.

Si une valeur d'argument **ByRef** est copiée dans la procédure et que l'argument et le paramètre sont du même type, aucune conversion n'est nécessaire. Mais si les types sont différents, Visual Basic doit effectuer une conversion dans les deux sens. Si l'un des types est votre type de classe ou de structure, Visual Basic doit le convertir vers et à partir de l'autre type. Cela signifie que vous devez définir des opérateurs de conversion dans les deux sens.

ID d'erreur : BC33037

Pour corriger cette erreur

- Si possible, utilisez un argument d'appel du même type que le paramètre de procédure. Ainsi, Visual Basic n'a pas besoin d'effectuer une conversion.
- Si vous devez appeler la procédure avec un type d'argument autre que le type du paramètre, sans retourner une valeur dans l'argument d'appel, affectez au paramètre la valeur ByVal au lieu de **ByRef**.
- Si vous devez retourner une valeur dans l'argument d'appel, définissez l'opérateur de conversion inverse afin que Visual Basic puisse convertir le type en type d'argument d'appel.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Procédures d'opérateur

Impossible de créer une instance de Module '<nommodule>'

Cannot create an instance of Module '<modulename>'

Un module existe uniquement en tant qu'instance partagée unique et il n'est pas possible de créer des instances supplémentaires.

ID d'erreur : BC30166

Pour corriger cette erreur

- Remplacez le module par une classe ou remplacez-le par un nom de classe dans la clause **New**.

Voir aussi

Référence

Module, instruction

Concepts

Implements, mot clé et instruction

Impossible de trouver le répertoire .NET Framework : <erreur>

Cannot find .NET Framework directory: <error>

Le répertoire contenant les assemblys du .NET Framework ne peut pas être localisé.

ID d'erreur : BC31508

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si cette erreur persiste, enregistrez votre travail et redémarrez Visual Studio.
3. Si l'erreur se reproduit, réinstallez Visual Basic.
4. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Impossible d'implémenter '<NomInterface1>.<NomMembre>', car son implémentation pourrait être en conflit avec l'implémentation de '<NomInterface2>.<NomMembre>' pour certains arguments de type

Cannot implement '<interfacename1>.<membername>' because its implementation could conflict with the implementation for '<interfacename2>.<membername>' for some type arguments

Une classe implémentant plusieurs interfaces génériques, l'une héritant de l'autre, deux implémentations d'un membre d'interface pourraient être en conflit pour certaines valeurs d'arguments de type.

Les instructions suivantes peuvent générer cette erreur.

```
Public Interface iFace1(Of t)
    Sub testSub()
End Interface
Public Interface iFace2(Of u)
    Inherits iFace1(Of u)
End Interface
Public Class testClass(Of y, z)
    Implements iFace1(Of y), iFace2(Of z)
    Public Sub testSuby() Implements iFace1(Of y).testSub
    End Sub
    Public Sub testSubz() Implements iFace1(Of z).testSub
    End Sub
End Class
```

`iFace2` héritant d'`iFace1` à l'aide de son propre paramètre de type (`u`), `testClass` implémenterait deux versions d'`iFace1.testSub` avec des signatures identiques si le même argument de type était passé à `y` et `z`. Cela produirait une ambiguïté à propos de la version à laquelle accéder.

ID d'erreur : BC32125

Pour corriger cette erreur

- Modifiez la structure d'héritage des interfaces, afin que `iFace1` ne puisse pas être implémentée avec deux arguments de type différents.

  - ou -

- Supprimez de l'instruction **Implements** l'une des interfaces provoquant le conflit d'implémentation.

Voir aussi

Référence

Class, instruction (Visual Basic)
Interface, instruction (Visual Basic)
Implements, instruction

Concepts

Implements, mot clé et instruction
Types génériques en Visual Basic

Impossible d'implémenter l'interface '<NomInterface1>', car son implémentation pourrait être en conflit avec l'implémentation d'une autre interface '<NomInterface2>' implémentée pour certains arguments de type

Cannot implement interface '<interfacename1>' because its implementation could conflict with the implementation of another implemented interface '<interfacename2>' for some type arguments

Une déclaration de classe contient une instruction **Implements** qui spécifie deux interfaces ou plus, mais au moins l'une des interfaces est générique et deux des implémentations peuvent entrer en conflit pour certaines valeurs d'arguments de type.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Public Interface iFace1
    Sub testSub(ByVal arg As String)
End Interface
Public Interface iFace2(Of t)
    Sub testSub(ByVal arg As t)
End Interface
Public Class testClass
    Implements iFace1, iFace2(Of String)
End Class
```

`iFace2` étant construit à l'aide de **String**, `testClass` doit implémenter deux versions de `testSub` avec des signatures identiques. Cela crée une ambiguïté sur la version à accéder.

ID d'erreur : BC32072

Pour corriger cette erreur

- Modifiez l'argument de type fourni à l'interface générique pour éviter tout conflit.

  - ou -

- Supprimez de l'instruction **Implements** l'une des interfaces qui provoquent le conflit d'implémentation.

Voir aussi

Référence

Class, instruction (Visual Basic)
Interface, instruction (Visual Basic)
Implements, instruction

Concepts

Implements, mot clé et instruction
Types génériques en Visual Basic

Impossible d'hériter de l'interface '<NomInterface1>', car elle pourrait être identique à l'interface '<NomInterface2>' pour certains arguments de type

Cannot inherit interface '<interfacename1>' because it could be identical to interface '<interfacename2>' for some type arguments

Une interface générique hérite plusieurs fois d'une autre interface générique, et deux des héritages pourraient être en conflit pour certaines valeurs d'arguments de type.

Les instructions suivantes peuvent générer cette erreur.

```
Public Interface interfaceA(Of u)

End Interface

Public Interface derivedInterface(Of t1, t2)

Inherits interfaceA(Of t1), interfaceA(Of t2)

End Interface
```

Si l'`derivedInterface` est construite ou implémentée en fournissant le même type à `t1` et `t2`, elle doit hériter de deux versions de l'`interfaceA` avec des arguments de type identiques. Cela produirait une ambiguïté à propos de la version à laquelle accéder.

ID d'erreur : BC32120

Pour corriger cette erreur

- Modifiez l'un des arguments de type fournis à l'interface dérivée afin qu'il n'y ait aucun conflit.

  - ou -

- Supprimez de l'instruction **Inherits** l'une des interfaces qui provoquent l'héritage potentiel ou le conflit d'implémentation.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Inherits, instruction

Concepts

Vue d'ensemble des interfaces
Éléments fondamentaux de l'héritage
Types génériques en Visual Basic

Impossible d'hériter de l'interface '<NomInterface1>', car elle pourrait être identique à l'interface '<NomInterface2>' dont l'interface '<NomInterface3>' hérite pour certains arguments de type

Cannot inherit interface '<interfacename1>' because it could be identical to interface '<interfacename2>' from which the interface '<interfacename3>' inherits for some type arguments

Une interface générique hérite d'au moins deux interfaces génériques, et deux des héritages peuvent entrer en conflit pour certaines valeurs d'arguments de type.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Public Interface interfaceA(Of u)
End Interface
Public Interface interfaceX(Of v)
    Inherits interfaceA(Of v)
End Interface
Public Interface derivedInterface(Of t1, t2)
    Inherits interfaceA(Of t1), interfaceX(Of t2)
End Interface
```

Si `derivedInterface` est construit ou implémenté en fournissant le même type à `t1` et `t2`, il doit hériter de deux versions de `interfaceA` avec des arguments de type identiques. Cela crée une ambiguïté sur la version à accéder.

ID d'erreur : BC32123

Pour corriger cette erreur

- Modifiez l'un des arguments de type fournis à l'interface dérivée pour éviter tout conflit.

  - ou -

- Supprimez de l'instruction **Inherits** l'une des interfaces qui provoquent le conflit d'héritage ou d'implémentation potentiel.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Inherits, instruction

Concepts

Vue d'ensemble des interfaces
Éléments fondamentaux de l'héritage
Types génériques en Visual Basic

Impossible d'hériter de l'interface '<NomInterface1>' car l'interface '<NomInterface2>' dont elle hérite peut être identique à l'interface '<NomInterface3>' pour certains arguments de type

Cannot inherit interface '<interfacename1>' because the interface '<interfacename2>' from which it inherits could be identical to interface '<interfacename3>' for some type arguments

Une interface générique hérite d'au moins deux interfaces génériques, et deux des héritages peuvent entrer en conflit pour certaines valeurs d'arguments de type.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Public Interface interfaceA(Of u)
    Inherits interfaceX(Of u)
End Interface
Public Interface interfaceX(Of v)
End Interface
Public Interface derivedInterface(Of t1, t2)
    Inherits interfaceA(Of t1), interfaceX(Of t2)
End Interface
```

Si `derivedInterface` est construit ou implémenté en fournissant le même type à `t1` et `t2`, il doit hériter de deux versions de `interfaceX` avec des arguments de type identiques. Cela crée une ambiguïté sur la version à accéder.

ID d'erreur : BC32121

Pour corriger cette erreur

- Modifiez l'un des arguments de type fournis à l'interface dérivée pour éviter tout conflit.

  - ou -

- Supprimez de l'instruction **Inherits** l'une des interfaces qui provoquent le conflit d'héritage ou d'implémentation potentiel.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Inherits, instruction

Concepts

Vue d'ensemble des interfaces
Éléments fondamentaux de l'héritage
Types génériques en Visual Basic

Impossible d'hériter de l'interface '<NomInterface1>', car l'interface '<NomInterface2>' dont elle hérite peut être identique à l'interface '<NomInterface3>' dont l'interface '<NomInterface4>' hérite pour certains arguments de type

Cannot inherit interface '<interfacename1>' because the interface '<interfacename2>' from which it inherits could be identical to interface '<interfacename3>' from which the interface '<interfacename4>' inherits for some type arguments

Une interface générique hérite d'au moins deux interfaces génériques, et deux des héritages peuvent entrer en conflit pour certaines valeurs d'arguments de type.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Public Interface interfaceA(Of u)
End Interface
Public Interface interfaceX(Of v)
    Inherits interfaceA(Of v)
End Interface
Public Interface interfaceY(Of w)
    Inherits interfaceA(Of w)
End Interface
Public Interface derivedInterface(Of t1, t2)
    Inherits interfaceX(Of t1), interfaceY(Of t2)
End Interface
```

Si `derivedInterface` est construit ou implémenté en fournissant le même type à `t1` et `t2`, il doit hériter de deux versions de `interfaceA` avec des arguments de type identiques. Cela crée une ambiguïté sur la version à accéder.

ID d'erreur : BC32122

Pour corriger cette erreur

- Modifiez l'un des arguments de type fournis à l'interface dérivée pour éviter tout conflit.

  - ou -

- Supprimez de l'instruction **Inherits** l'une des interfaces qui provoquent le conflit d'héritage ou d'implémentation potentiel.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Inherits, instruction

Concepts

Vue d'ensemble des interfaces
Éléments fondamentaux de l'héritage
Types génériques en Visual Basic

Impossible de faire référence à '<nom>', car il est membre du champ de valeurs '<nom>' de la classe '<nomclasse>', qui a 'System.MarshalByRefObject' comme classe de base

Cannot refer to '<name>' because it is a member of the value-typed field '<name>' of class '<classname>' which has 'System.MarshalByRefObject' as a base class

La classe **System.MarshalByRefObject** active des applications qui prennent en charge l'accès distant à des objets au-delà des limites de domaine d'application. Les types doivent hériter de la classe **MarshalByRejectObject** lorsque le type est utilisé au-delà des limites de domaine d'application. L'état de l'objet ne doit pas être copié car les membres de l'objet ne peuvent pas être utilisés en dehors du domaine d'application dans lequel ils ont été créés.

ID d'erreur : BC30310

Pour corriger cette erreur

1. Vérifiez la référence afin de vous assurer que le membre auquel il est fait référence est valide.
2. Qualifiez de manière explicite le membre à l'aide du mot clé **Me**.

Voir aussi

Référence

Dim, instruction (Visual Basic)
MarshalByRefObject

Impossible de faire référence à un membre instance d'une classe à partir d'une méthode partagée ou d'un initialiseur de membre partagé sans une instance explicite de la classe

Cannot refer to an instance member of a class from within a shared method or shared member initializer without an explicit instance of the class

Vous avez essayé de faire référence au membre non partagé d'une classe à partir d'une procédure partagée. L'exemple suivant illustre une telle situation.

```
Class sample
    Public x as Integer
    Public Shared Sub setX()
        x = 10
    End Sub
End Class
```

Dans l'exemple précédent, l'instruction d'assignation `x = 10` génère ce message d'erreur. Cela est dû au fait qu'une procédure partagée tente d'accéder à une variable d'instance.

La variable `x` est un membre d'instance parce qu'elle n'est pas déclarée en tant que Shared (Visual Basic). Chaque instance de la classe `sample` contient sa propre variable individuelle `x`. Lorsqu'une instance définit ou modifie la valeur de `x`, elle n'affecte la valeur de `x` dans aucune autre instance.

Toutefois, la procédure `setX` est **Shared** entre toutes les instances de la classe `sample`. Cela signifie qu'elle n'est associée à aucune instance de la classe, mais qu'elle fonctionne plutôt indépendamment des instances individuelles. N'ayant pas de connexion avec une instance particulière, la procédure `setX` ne peut pas accéder à une variable d'instance. Elle ne doit fonctionner que sur les variables **Shared**. Lorsque `setX` définit ou modifie la valeur d'une variable partagée, la nouvelle valeur est accessible à toutes les instances de la classe.

ID d'erreur : BC30369

Pour corriger cette erreur

1. Déterminez si vous souhaitez partager le membre entre toutes les instances de la classe ou en laisser une copie individuelle pour chaque instance.
2. Si vous souhaitez qu'une seule copie du membre soit partagée entre toutes les instances, ajoutez le mot clé **Shared** à la déclaration de membre. Conservez le mot clé **Shared** dans la déclaration de procédure.
3. Si vous souhaitez que chaque instance possède sa propre copie individuelle du membre, ne spécifiez pas **Shared** dans la déclaration de membre. Supprimez le mot clé **Shared** de la déclaration de procédure.

Voir aussi

Référence

Shared (Visual Basic)

Impossible de spécifier à la fois /win32icon et /win32resource

Cannot specify both /win32icon and /win32resource

Les options **/win32resource** et **/win32icon** sont mutuellement exclusives, car toutes deux insèrent des informations d'icônes dans le fichier de sortie.

ID d'erreur : BC2023

Pour corriger cette erreur

- Utilisez uniquement l'option **/win32icon** pour insérer un fichier .ico dans le fichier de sortie. Ce fichier .ico représente le fichier de sortie dans l'Explorateur Windows.

  - ou -

- Utilisez uniquement l'option **/win32resource** pour insérer un fichier de ressources Win32 dans le fichier de sortie. Une ressource Win32 peut contenir des informations de version ou des bitmaps (icônes) qui permettent d'identifier facilement votre application dans l'Explorateur Windows.

Voir aussi

Référence

/win32icon
/win32resource

Impossible de définir la valeur d'une variable locale pour une méthode qui ne se trouve pas au sommet de la pile

Cannot set the value of a local variable for a method that is not at the top of the stack

Vous ne pouvez modifier que les variables qui se trouvent en haut de la pile des appels. Par exemple, si `procedure1` appelle `procedure2` et que vous vous trouvez dans `procedure1`, vous ne pouvez pas modifier les variables de `procedure2`.

ID d'erreur : BC30711

Pour corriger cette erreur

- Modifiez les variables qui se trouvent en haut de la pile des appels.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Impossible d'ouvrir '<nomfichier>' en écriture

Can't open '<filename>' for writing

Le fichier spécifié ne peut pas être ouvert pour écriture, peut-être parce qu'il a déjà été ouvert.

ID d'erreur : BC2012

Pour corriger cette erreur

1. Fermez le fichier, puis rouvrez-le.
2. Vérifiez les autorisations du fichier.

Voir aussi

Référence

My.Computer.FileSystem.WriteAllText, méthode
My.Computer.FileSystem.WriteAllBytes, méthode

Autres ressources

Écriture dans des fichiers en Visual Basic

Les instructions 'Case' ne sont pas valides dans la fenêtre Exécution

'Case' statements are not valid in the Immediate window

Les instructions **Case** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30719

Pour corriger cette erreur

- Supprimez les instructions **Case** de la fenêtre **Exécution**.

Voir aussi

Autres ressources

Débogage dans Visual Studio

'Case Else' ne peut apparaître qu'à l'intérieur d'une instruction 'Select Case'

'Case Else' can only appear inside a 'Select Case' statement

Une instruction **Case Else** apparaît à l'extérieur d'un bloc **Select**. Une instruction **Case Else** ne peut être utilisée qu'entre une instruction **Select** ou **Select Case** et son instruction **End Select** correspondante.

ID d'erreur : BC30071

Pour corriger cette erreur

- Supprimez l'instruction **Case Else** ou déplacez-la à l'intérieur d'un bloc **Select**.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

'Case' ne peut pas suivre 'Case Else' dans la même instruction 'Select'

'Case' cannot follow a 'Case Else' in the same 'Select' statement

Une instruction **Case Else** introduit des instructions à exécuter si aucune correspondance n'est trouvée pour l'instruction **Case** initiale. Une instruction **Case** a été rencontrée après **Case Else** dans le même bloc **Select**.

ID d'erreur : BC30321

Pour corriger cette erreur

- Déplacez **Case Else** vers l'emplacement approprié après l'instruction **Case**.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

'Case' ne peut apparaître qu'à l'intérieur d'une instruction 'Select Case'

'Case' can only appear inside a 'Select Case' statement

Une instruction **Case** apparaît à l'extérieur d'un bloc **Select**. Une instruction **Case** ne peut être utilisée qu'entre une instruction **Select** ou **Select Case** et son instruction **End Select** correspondante.

ID d'erreur : BC30072

Pour corriger cette erreur

- Supprimez l'instruction **Case** ou déplacez-la à l'intérieur d'un bloc **Select**.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

La casse du nom de l'espace de noms '<NomEspaceDeNoms1>' ne correspond pas à la casse du nom de l'espace de noms '<NomEspaceDeNoms2>' dans le fichier '<CheminFichier>'

Casing of namespace name '<namespacename1>' does not match casing of namespace name '<namespacename2>' in file '<filepath>'

Un espace de noms apparaît plusieurs fois dans le projet, mais avec des casses différentes.

La *casse* fait référence à l'utilisation de caractères en majuscules et en minuscules dans le nom d'un élément de programmation. Visual Basic ne respecte pas la casse, contrairement au Common Language Runtime (CLR). Pour plus d'informations, consultez « Respect de la casse dans les noms » dans Noms d'éléments déclarés.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40055

Pour corriger cette erreur

- Par mesure de précaution, utilisez toujours une casse identique dans chaque référence à un espace de noms. Cela permet d'éviter tout risque de confusion par le Common Language Runtime.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic
Noms d'éléments déclarés
Conventions d'affectation de noms Visual Basic

Le bloc 'Catch' n'a jamais été atteint, car '<nom1>' hérite de '<nom2>'

'Catch' block never reached, because '<name1>' inherits from '<name2>'

Un bloc **Catch** dans le code n'est jamais atteint.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42029

Pour corriger cette erreur

- Supprimez le bloc **Catch** superflu.

Voir aussi

Tâches

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Référence

Try...Catch...Finally, instruction (Visual Basic)

Bloc 'Catch' jamais atteint ; <exception> géré au-dessus dans la même instruction 'Try'

'Catch' block never reached; <exception> handled above in the same 'Try' statement

Un bloc **Catch** du code ne peut pas être atteint car il est géré dans un bloc **Try** précédent.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42031

Pour corriger cette erreur

- Supprimez l'instruction redondante.

Voir aussi

Tâches

Comment : intercepter une exception dans Visual Basic
Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic
Comment : filtrer des erreurs dans un bloc Catch dans Visual Basic
Procédure pas à pas : gestion structurée des exceptions

Référence

Try...Catch...Finally, instruction (Visual Basic)

'Catch' ne peut pas apparaître après 'Finally' dans une instruction 'Try'

'Catch' cannot appear after 'Finally' within a 'Try' statement

Une instruction **Catch** apparaît dans le code après **Finally** qui termine un bloc d'instructions **Try**. **Catch** doit apparaître dans un bloc d'instructions **Try...Catch...Finally**.

ID d'erreur : BC30379

Pour corriger cette erreur

- Déplacez l'instruction **Catch** vers un emplacement plus approprié dans le code.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'Catch' ne peut pas apparaître en dehors d'une instruction 'Try'

'Catch' cannot appear outside a 'Try' statement

Catch doit apparaître dans un bloc d'instructions **Try...Catch...Finally**. Vous disposez d'une instruction **Catch** inutile dans votre bloc **Try** ou votre instruction **Catch** apparaît en dehors des limites de son bloc **Try** correspondant.

ID d'erreur : BC30380

Pour corriger cette erreur

- Supprimez l'instruction **Catch** si elle n'est pas nécessaire ou placez-la dans un bloc d'instructions **Try...Catch...Finally**.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'Catch' ne peut pas intercepter le type '<nomtype>', car il n'est pas 'System.Exception' ou une classe qui hérite de 'System.Exception'

'Catch' cannot catch type '<typename>' because it is not 'System.Exception' or a class that inherits from 'System.Exception'

Catch peut seulement intercepter des exceptions, et vous avez tenté d'intercepter un élément qui n'est pas dérivé d'une exception.

ID d'erreur : BC30392

Pour corriger cette erreur

- Supprimez l'instruction **Catch** ou remplacez sa cible par une exception réelle.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'Catch' doit se terminer par une 'End Try' correspondante

'Catch' must end with a matching 'End Try'

Une instruction **Catch** apparaît dans votre code sans une instruction **End Try** correspondante. Les instructions **Catch** doivent se terminer par une instruction **End Try**.

ID d'erreur : BC30441

Pour corriger cette erreur

1. Supprimez l'instruction **Catch**.
2. Ajoutez une instruction **End Try** pour terminer le bloc.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Les instructions 'Catch' ne sont pas valides dans la fenêtre Exécution

'Catch' statements are not valid in the Immediate window

Les instructions **Catch** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30715

Pour corriger cette erreur

- Supprimez les instructions **Catch** du code de débogage.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Débogage dans Visual Studio

Impossible de convertir les valeurs 'Char' en '<nomtype>'

Message d'erreur
Impossible de convertir les valeurs 'Char' en '<nomtype>'. Utilisez soit Microsoft.VisualBasic.AscW pour interpréter un caractère comme une valeur Unicode, soit Microsoft.VisualBasic.Val pour interpréter le caractère comme un chiffre.
'Char' values cannot be converted to '<typename>'. Use Microsoft.VisualBasic.AscW to interpret a character as a Unicode value or Microsoft.VisualBasic.Val to interpret it as a digit.

Une expression tente de convertir une valeur **Char** en un type de données autre que **String** ou **Object**.

ID d'erreur : BC32006

Pour corriger cette erreur

- Utilisez la fonction **AscW** pour interpréter une valeur **Char** comme code de caractère Unicode ou la fonction **Val** pour l'interpréter comme un chiffre.

Voir aussi

Référence
Fonctions Asc, AscW
Val, fonction
Char, type de données (Visual Basic)

Une constante caractère doit contenir exactement un caractère

Character constant must contain exactly one character

Une constante caractère n'est pas valide car soit elle contient plusieurs caractères, soit c'est un caractère Null (tous les bits ont la valeur zéro).

ID d'erreur : BC30004

Pour corriger cette erreur

- Remplacez la constante caractère par un caractère unique ou remplacez le caractère Null par un caractère valide.

Voir aussi

Référence

Char, type de données (Visual Basic)

Caractère non valide

Character is not valid

Le contexte en cours n'autorise pas ce caractère. Par exemple, un tilde (**~**) ou une barre verticale (**|**) ne peut pas faire partie d'un nom d'élément déclaré.

ID d'erreur : BC30037

Pour corriger cette erreur

1. Vérifiez si un caractère supplémentaire a été accidentellement inséré dans l'élément de programmation.
2. Assurez-vous que l'élément de programmation est orthographié ou construit uniquement à l'aide de caractères valides pour ce type d'élément.

Voir aussi

Concepts

Noms d'éléments déclarés

La classe '<NomClasse>' n'a pas de 'Sub New' accessible et ne peut pas être héritée.

Class '<classname>' has no accessible 'Sub New' and cannot be inherited

Une classe utilise l'instruction Inherits pour spécifier une classe de base, mais elle ne peut pas accéder aux constructeurs de la classe de base prévue.

Cela peut se produire si la classe de base prévue ne contient pas de constructeurs ou si elle possède des constructeurs dont les niveaux d'accès empêchent l'accès d'une autre classe.

Lorsque vous héritez d'une classe, votre constructeur doit appeler le constructeur de la classe de base à l'aide de MyBase. Si vous n'effectuez pas cet appel, ou si vous n'écrivez pas à un constructeur explicite, Visual Basic génère un constructeur implicite qui appelle `MyBase.New()`.

ID d'erreur : BC31399

Pour corriger cette erreur

1. Si vous disposez d'un contrôle de code source sur la classe de base prévue, modifiez le niveau d'accès d'au moins un de ses constructeurs afin qu'une autre classe puisse y accéder.
2. Si vous ne pouvez pas modifier les niveaux d'accès des constructeurs de la classe de base prévue, héritez d'une autre classe ou n'héritez pas du tout.

Voir aussi

Référence

Inherits, instruction
MyBase
New (Visual Basic)

Concepts

Éléments fondamentaux de l'héritage
Niveaux d'accès dans Visual Basic

La classe '<nomclasse>' est introuvable

Message d'erreur
La classe '<nomclasse>' est introuvable. Cette condition résulte généralement d'un fichier 'Microsoft.VisualBasic.dll' incompatible.
Class '<classname>' cannot be found. This condition is usually the result of a mismatched 'Microsoft.VisualBasic.dll'.

Un membre défini n'a pas pu être trouvé.

ID d'erreur : BC31098

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si l'erreur se reproduit, enregistrez votre travail et redémarrez Visual Studio.
3. Si l'erreur persiste, réinstallez Visual Basic.
4. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources
Support technique et accessibilité

La classe '<nomclasse>' ne peut pas être indexée, car elle n'a pas de propriété par défaut

Class '<classname>' cannot be indexed because it has no default property

Vous n'avez pas réussi à spécifier une propriété par défaut pour une classe.

ID d'erreur : BC30367

Pour corriger cette erreur

- Spécifiez une propriété comme la propriété par défaut en faisant débuter sa déclaration par le mot clé **Default**.

Voir aussi

Référence

Default (Visual Basic)

Concepts

Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0

La classe '<NomClasse>' ne peut pas hériter d'elle-même : <message>

Class '<classname>' cannot inherit from itself: <message>

Une instruction Inherits dans une définition de classe spécifie sa propre classe.

Une classe peut hériter d'une autre classe, qui lui fournit tous les membres de la classe dont elle hérite. Par conséquent, elle ne doit pas redéfinir ces membres. Une telle classe est appelée *classe dérivée*, et la classe dont elle hérite est appelée *classe de base*.

Cela n'a pas sens qu'une classe hérite d'elle-même, parce qu'elle possède déjà tous ses propres membres.

ID d'erreur : BC30257

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom de la classe dans l'instruction **Inherits**.
2. Si vous ne comptez pas hériter d'une autre classe, supprimez l'intégralité de l'instruction **Inherits**.
3. Examinez le message cité pour obtenir des suggestions.

Voir aussi

Autres ressources

Héritage en Visual Basic
Présentation des classes

La classe '<nomclasse>' n'a pas pu être créée : <erreur>

Class '<classname>' could not be created: <error>

Une initialisation de classe a échoué. L'une des causes possibles est une installation du .NET Framework endommagée.

ID d'erreur : BC32400

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Réinstallez le .NET Framework et recompilez le programme.
3. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
4. Si l'erreur persiste après la réinstallation, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

La contrainte 'Class' ne peut pas être spécifiée plusieurs fois pour le même paramètre de type

'Class' constraint cannot be specified multiple times for the same type parameter

Une liste de contraintes inclut plusieurs fois la contrainte Class (Visual Basic).

Une liste de contraintes d'un paramètre de type peut spécifier que l'argument de type passé à ce paramètre de type doit être un type valeur (avec la contrainte Structure (Visual Basic)) ou un type référence (avec la contrainte **Class**). Vous ne pouvez pas spécifier les deux contraintes pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32101

Pour corriger cette erreur

- Supprimez tous les mots clés **Class** redondants. Vous ne devez en avoir qu'un seul dans la liste de contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

La contrainte 'Class' et la contrainte 'Structure' ne peuvent pas être combinées

'Class' constraint and 'Structure' constraint cannot be combined

Une liste de contraintes inclut à la fois la contrainte Class (Visual Basic) et la contrainte Structure (Visual Basic).

Une liste de contraintes d'un paramètre de type peut spécifier que l'argument de type passé à ce paramètre de type doit être un type valeur (avec la contrainte **Structure**) ou un type référence (avec la contrainte **Class**). Vous ne pouvez pas spécifier les deux contraintes pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32104

Pour corriger cette erreur

- Déterminez s'il est nécessaire d'avoir un type valeur ou un type référence pour l'argument de type.
    - Si vous souhaitez que l'argument de type soit un type valeur, supprimez le mot clé **Class** de la liste de contraintes.
    - Si vous souhaitez que l'argument de type soit un type référence, supprimez le mot clé **Structure** de la liste de contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

Une contrainte 'Class' et une contrainte de type classe spécifique ne peuvent pas être combinées

'Class' constraint and a specific class type constraint cannot be combined

Une liste de contraintes contient la contrainte Class (Visual Basic) et le nom d'une classe définie.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Vous pouvez spécifier les éléments requis suivants selon n'importe quelle combinaison :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit hériter d'une classe au plus
- L'argument de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder (inclure la contrainte **New**)

Si la liste des contraintes ne contient pas de classes ou d'interfaces spécifiques, vous pouvez imposer une exigence plus générale en spécifiant l'un des éléments suivants :

- L'argument de type doit être un type valeur (inclure la contrainte **Structure**)
- L'argument de type doit être un type référence (inclure la contrainte **Class**)

Vous ne pouvez pas spécifier **Structure** et **Class** pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32107

Pour corriger cette erreur

- Pour que l'argument de type soit d'un type référence, supprimez le nom de la classe de la liste des contraintes.
- Pour que l'argument de type hérite du nom de classe spécifié, supprimez le mot clé **Class** de la liste des contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

La classe '<NomClasse1>' doit déclarer un 'Sub New', car sa classe de base '<NomClasse2>' comporte plusieurs 'Sub New' accessibles qu'il est possible d'appeler sans argument.

Class '<classname1>' must declare a 'Sub New' because its base class '<classname2>' has more than one accessible 'Sub New' that can be called with no arguments

Une classe dérivée ne déclare pas de constructeur, et Visual Basic ne peut pas en générer un car il ne peut pas déterminer le constructeur de classe de base à appeler.

Lorsqu'une classe dérivée ne déclare pas de constructeur, Visual Basic tente de générer un constructeur sans paramètre implicite qui appelle **MyBase.New()**. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, ou s'il en existe plusieurs, Visual Basic ne peut pas générer un constructeur implicite.

Cette situation peut se produire, par exemple, si un constructeur de classe de base a un argument **Optional** unique et qu'un autre à un argument **ParamArray** unique. Chacun de ces constructeurs peut être appelé sans argument.

ID d'erreur : BC32036

Pour corriger cette erreur

1. Déclarez et implémentez au moins un constructeur **Sub New** dans la classe dérivée.
2. Ajoutez un appel à un constructeur de classe de base, **MyBase.New()**, sur la première ligne de chaque **Sub New**.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs
Optional (Visual Basic)
ParamArray

Concepts

Durée de vie d'un objet : création et destruction des objets
Paramètres facultatifs
Tableaux de paramètres

La classe '<NomClasse>' doit déclarer un 'Sub New', car le '<NomConstructeur>' dans sa classe de base '<NomClasseDeBase>' est marqué comme obsolète : '<MessageErreur>'

Class '<classname>' should declare a 'Sub New' because the '<constructorname>' in its base class '<baseclassname>' is marked obsolete: '<errormessage>'

Une déclaration de classe ne contient pas de constructeur, et le constructeur de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme un avertissement.

Lorsqu'une classe dérivée ne déclare pas de constructeur, Visual Basic tente de générer un constructeur sans paramètre implicite qui appelle **MyBase.New()**. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un constructeur implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41002

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Utilisez **Sub New** pour déclarer un constructeur dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La classe '<NomClasse>' doit déclarer un 'Sub New', car le '<NomConstructeur>' de sa classe de base '<NomClasseDeBase>' est marqué comme obsolete

Class '<classname>' should declare a 'Sub New' because the '<constructorname>' in its base class '<baseclassname>' is marked obsolete

Une déclaration de classe ne contient pas de constructeur, et le constructeur de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme un avertissement.

Lorsqu'une classe dérivée ne déclare pas de constructeur, Visual Basic tente de générer un constructeur sans paramètre implicite qui appelle **MyBase.New()**. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un constructeur implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41001

Pour corriger cette erreur

- Utilisez **Sub New** pour déclarer un constructeur dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La classe '<nomclasse>' doit être déclarée 'MustInherit' ou se substituer aux membres 'MustOverride' hérités suivants : <nommembre>

Class '<classname>' must either be declared 'MustInherit' or override the following inherited 'MustOverride' member(s): <membername(s)>

Les classes dérivées de classes de base contenant des membres **MustOverride** doivent soit se substituer à ces membres, soit utiliser le modificateur **MustInherit**.

ID d'erreur : BC30610

Pour corriger cette erreur

- Ajoutez le modificateur **MustInherit** à la définition de classe.
- Déclarez une substitution à l'aide du mot clé **Overrides**.

Voir aussi

Référence

Overrides

MustInherit

Autres ressources

Héritage en Visual Basic

La classe '<NomClasse>' doit déclarer un 'Sub New', car le '<NomConstructeur>' dans sa classe de base '<NomClasseDeBase>' est marqué comme obsolète : '<MessageErreur>'

Class '<classname>' must declare a 'Sub New' because the '<constructorname>' in its base class '<baseclassname>' is marked obsolete: '<errormessage>'

Une déclaration de classe ne contient pas de constructeur, et le constructeur de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme une erreur.

Lorsqu'une classe dérivée ne déclare pas de constructeur, Visual Basic tente de générer un constructeur sans paramètre implicite qui appelle **MyBase.New()**. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un constructeur implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30918

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Utilisez **Sub New** pour déclarer un constructeur dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La classe '<NomClasse>' doit déclarer un 'Sub New', car le '<NomConstructeur>' dans sa classe de base '<NomClasseDeBase>' est marqué comme obsolète.

Class '<classname>' must declare a 'Sub New' because the '<constructorname>' in its base class '<baseclassname>' is marked obsolete

Une déclaration de classe ne contient pas de constructeur, et le constructeur de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme une erreur.

Lorsqu'une classe dérivée ne déclare pas de constructeur, Visual Basic tente de générer un constructeur sans paramètre implicite qui appelle **MyBase.New()**. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un constructeur implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30917

Pour corriger cette erreur

- Utilisez **Sub New** pour déclarer un constructeur dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La classe '<NomClasse>' doit déclarer un 'Sub New', car sa classe de base '<NomClasse2>' n'a pas de 'Sub New' accessible qu'il est possible d'appeler sans argument

Class '<classname>' must declare a 'Sub New' because its base class '<classname2>' does not have an accessible 'Sub New' that can be called with no arguments

Une classe dérivée ne déclare pas de constructeur, et Visual Basic ne peut pas en générer un parce qu'il n'y a aucun constructeur de classe de base qu'il peut appeler.

Lorsqu'une classe dérivée ne déclare pas de constructeur, Visual Basic tente de générer un constructeur sans paramètre implicite qui appelle **MyBase.New()**. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, ou s'il y en a plusieurs, Visual Basic ne peut pas générer de constructeur implicite.

ID d'erreur : BC30387

Pour corriger cette erreur

1. Déclarez et implémentez au moins un constructeur **Sub New** dans la classe dérivée.
2. Ajoutez un appel à un constructeur de classe de base, **MyBase.New()**, sur la première ligne de chaque **Sub New**.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs
Optional (Visual Basic)
ParamArray

Concepts

Durée de vie d'un objet : création et destruction des objets
Paramètres facultatifs
Tableaux de paramètres

Les classes qui sont génériques ou contenues dans un type générique ne peuvent pas hériter d'une classe d'attributs

Classes that are generic or contained in a generic type cannot inherit from an attribute class

Une classe qui est générique ou imbriquée dans un type générique spécifie qu'elle hérite d'une classe d'attributs.

Visual Basic et le .NET Framework ne prennent pas en charge actuellement les combinaisons d'attributs et de types génériques. Cela signifie les restrictions suivantes s'appliquent :

- Un attribut ne peut pas être un type générique ou être déclaré dans un type générique.
- Un attribut ne peut pas hériter d'une classe générique, et inversement.
- Lorsque vous appliquez un attribut, vous ne pouvez pas fournir un argument qui fait partie des éléments suivants :
    - un type générique,
    - un type construit à partir d'un type générique,
    - un paramètre de type d'un type conteneur, ou
    - un type construit à partir d'un paramètre de type d'un type conteneur.

ID d'erreur : BC32074

Pour corriger cette erreur

- Remplacez la classe de base par une classe autre qu'une classe d'attributs, ou supprimez complètement l'instruction **Inherits**.

Voir aussi

Référence

Inherits, instruction
Attribute

Concepts

Vue d'ensemble des attributs dans Visual Basic
Types génériques en Visual Basic
Éléments fondamentaux de l'héritage

Les classes ne peuvent pas être déclarées '<spécificateur>'

Classes cannot be declared '<specifier>'

Vous avez tenté de déclarer **Class** ou **Module** avec un spécificateur non valide.

ID d'erreur : BC30461

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Class, instruction (Visual Basic)
Module, instruction

Les classes ne peuvent hériter que d'autres classes

Classes can inherit only from other classes

Une instruction **Inherits** spécifie comme type de base un type autre qu'une classe.

ID d'erreur : BC30258

Pour corriger cette erreur

- Remplacez le type de base par une classe déjà définie.

Voir aussi

Concepts

Éléments fondamentaux de l'héritage

L'événement 'Class_Terminate' n'est plus pris en charge

Message d'erreur
L'événement 'Class_Terminate' n'est plus pris en charge. Utilisez 'Sub Finalize' pour libérer des ressources.
'Class_Terminate' event is no longer supported. Use 'Sub Finalize' to free resources.

L'événement **Class_Terminate** des versions précédentes de Visual Basic a été remplacé par les destructeurs de classe.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42002

Pour corriger cette erreur

- Déclarez une procédure **Sub** nommée **Finalize** pour mettre fin à une classe. **Sub Finalize** est appelé lorsque le garbage collector ne détecte plus de références actives à l'instance.

Voir aussi

Tâches
Comment : implémenter le modèle DisposeFinalize (Visual Basic)
Référence
Utilisation de constructeurs et de destructeurs
Concepts
Classes pour les utilisateurs de Visual Basic 6.0

L'événement 'Class_Initialize' n'est plus pris en charge

Message d'erreur
L'événement 'Class_Initialize' n'est plus pris en charge. Utilisez 'Sub New' pour initialiser une classe.
'Class_Initialize' event is no longer supported. Use 'Sub New' to initialize a class.

L'événement **Class_Initialize** des versions précédentes de Visual Basic a été remplacé par les constructeurs de classe.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42001

Pour corriger cette erreur

- Déclarez une ou plusieurs procédures **Sub** nommées **New** pour initialiser une classe. **Sub New** est appelée lorsqu'une instance de classe a été récemment créée.

Voir aussi

Référence
Utilisation de constructeurs et de destructeurs
Concepts
Modifications de Class_Initialize pour les utilisateurs de Visual Basic 6.0
Classes pour les utilisateurs de Visual Basic 6.0

Une instruction 'Class' doit se terminer par une 'End Class' correspondante

'Class' statement must end with a matching 'End Class'

Class sert à initialiser un bloc **Class** ; elle ne peut dès lors apparaître qu'au début du bloc, avec une instruction**End Class** correspondante qui termine le bloc. Vous possédez une instruction **Class** redondante ou vous n'avez pas terminé votre bloc **Class** par **End Class**.

ID d'erreur : BC30481

Pour corriger cette erreur

- Recherchez et supprimez l'instruction **Class** inutile.
- Terminez le bloc **Class** par une instruction **End Class** correspondante.

Voir aussi

Référence

End, instruction

Une classe dans une interface ne peut pas être déclarée '<spécificateur>'

Class in an interface cannot be declared '<specifier>'

Un modificateur non valide a été spécifié avec une classe déclarée dans une interface.

ID d'erreur : BC31070

Pour corriger cette erreur

- Supprimez les modificateurs, tels que **Public**, **Private**, **Shared**, **Friend**, **Protected**, **Protected Friend** ou **Overridable**.

Voir aussi

Référence

Class, instruction (Visual Basic)

Autres ressources

Interfaces en Visual Basic

La page de codes '<nom>' n'est pas valide ou n'est pas installée

Codepage '<name>' is invalid or not installed

La page de codes spécifiée n'est pas valide ou est introuvable.

ID d'erreur : BC2016

Pour corriger cette erreur

- Examinez votre code et déterminez la solution appropriée à votre cas.

Voir aussi

Autres ressources

Présentation du débogueur

Conflits avec '<nom1>', qui est implicitement créé pour '<nom2>' dans <type> '<nom3>'

Conflicts with '<name1>', which is implicitly created for '<name2>' in <type> '<name3>'

Le nom d'un membre de type est en conflit avec un autre membre implicitement créé.

ID d'erreur : BC31058

Pour corriger cette erreur

- Renommez le membre déclaré explicitement afin de supprimer le conflit de noms.

Voir aussi

Concepts

Instructions de déclaration en Visual Basic

Conflit entre la propriété par défaut et 'DefaultMemberAttribute' défini pour '|1'

Conflict between the default property and the 'DefaultMemberAttribute' defined on '|1'

Une classe, une structure, ou une interface déclare une propriété par défaut avec le mot clé Default (Visual Basic), mais applique également DefaultMemberAttribute pour désigner un autre membre comme membre par défaut.

Un type peut avoir, au plus, un membre par défaut. Lorsque vous déclarez une propriété par défaut, Visual Basic applique automatiquement **DefaultMemberAttribute** à la classe, la structure ou l'interface qui désigne cette propriété.

ID d'erreur : BC32304

Pour corriger cette erreur

1. Déterminez le membre qui doit être le membre par défaut de la classe, la structure ou l'interface.
2. Supprimez la déclaration en conflit (le mot clé **Default** ou **DefaultMemberAttribute**).

Voir aussi

Tâches

Comment : déclarer et appeler une propriété par défaut en Visual Basic

Référence

Default (Visual Basic)

DefaultMemberAttribute

Concepts

Propriétés par défaut

Échec de la compilation : <message>

Compilation failed: <message>

La compilation a échoué en raison des circonstances spécifiées dans le message d'erreur

ID d'erreur : BC2010

Pour corriger cette erreur

- Vérifiez le message d'erreur pour déterminer la cause précise de l'échec.

Voir aussi

Autres ressources

Compiling an Application

'Compare' attendu

'Compare' expected

Une instruction **Option** spécifie **Text** ou **Binary**, mais pas **Compare**.

ID d'erreur : BC30208

Pour corriger cette erreur

- Ajoutez le mot clé **Compare** à l'instruction **Option**, par exemple `Option Compare Binary`.

Voir aussi

Référence

Option Compare, instruction

Virgule, ')' ou continuation d'expression valide attendue

Comma, ')', or a valid expression continuation expected

Les membres d'une liste, telle qu'une liste d'arguments dans une définition de procédure ou un appel de procédure, ne sont pas correctement séparés ou le caractère de fin de liste n'est pas correct. Le fait que deux membres de liste soient séparés uniquement par un espace ou par des espaces peut être l'une des causes possibles.

ID d'erreur : BC32017

Pour corriger cette erreur

- Utilisez une virgule (,) pour séparer les membres de liste et une parenthèse fermante ()) pour terminer la liste.

Voir aussi

Concepts

Sub, procédures
Function, procédures

Virgule ou ')' attendue (liste d'arguments type)

Comma or ')' expected (Type Argument List)

Une liste d'arguments de type contient un élément inattendu. Utilisez des virgules (**,**) pour séparer les arguments de type dans ce type de liste et utilisez une parenthèse fermante (**)**) pour terminer la liste.

ID d'erreur : BC32094

Pour corriger cette erreur

- Ajoutez une virgule et continuez la liste d'arguments de type ou ajoutez un caractère de parenthèse fermante pour terminer la liste.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Virgule ou ')' attendue (liste de paramètres type)

Comma or ')' expected (Type Parameter List)

Une liste de paramètres de type contient un élément inattendu. Utilisez des virgules (**,**) pour séparer les paramètres de type de cette liste et utilisez une parenthèse fermante (**)**) pour terminer la liste.

ID d'erreur : BC32099

Pour corriger cette erreur

- Ajoutez une virgule et continuez la liste de paramètres de type, ou ajoutez une parenthèse fermante pour terminer la liste.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Virgule ou ')' attendue (liste des paramètres de procédure)

Comma or ')' expected (Procedure Parameter List)

Une liste de paramètres contient un élément inattendu. Utilisez des virgules (**,**) pour séparer les paramètres dans ce type de liste et utilisez une parenthèse fermante (**)**) pour terminer la liste.

ID d'erreur : BC30213

Pour corriger cette erreur

- Ajoutez une virgule et continuez la liste de paramètres de type ou ajoutez une parenthèse fermante pour terminer la liste.

Voir aussi

Tâches

Comment : définir un paramètre pour une procédure

Référence

Liste de paramètres

Virgule attendue

Comma expected

Une virgule (,) entre deux arguments manque dans la liste d'arguments.

ID d'erreur : BC30196

Pour corriger cette erreur

- Vérifiez que tous les arguments sont séparés par des virgules.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

Une déclaration Const ne peut pas avoir un initialiseur de tableau

Const declaration cannot have an array initializer

Vous avez tenté d'utiliser un initialiseur de tableau dans une déclaration **Const**.

ID d'erreur : BC30445

Pour corriger cette erreur

- Modifiez l'initialiseur en une expression valide en supprimant les accolades qui l'entourent.

Voir aussi

Référence

Const, instruction (Visual Basic)

La constante '<nomconstante>' ne peut pas dépendre de sa propre valeur

Constant '<constantname>' cannot depend on its own value

Vous avez créé une dépendance circulaire dans votre code, où une constante dépend de sa propre valeur. Par exemple, `Const a = Const b; Const b = Const a`.

ID d'erreur : BC30500

Pour corriger cette erreur

- Vérifiez votre code pour déterminer l'endroit où la constante est évaluée et modifiez-le en conséquence.

Voir aussi

Concepts

Vue d'ensemble des constantes

Une constante ne peut pas être la cible d'une assignation

Constant cannot be the target of an assignment

Une constante apparaît dans un contexte qui lui assigne une valeur. Seuls les variables, propriétés et éléments de tableaux accessibles en écriture peuvent avoir des valeurs qui leur sont assignées pendant l'exécution.

ID d'erreur : BC30074

Pour corriger cette erreur

- Remplacez la constante par une variable, une propriété ou un élément de tableau unique accessible en écriture.

Voir aussi

Concepts

Instructions d'assignation

Autres ressources

Constantes et énumérations en Visual Basic

Une expression constante est requise

Constant expression is required

Une instruction **Const** n'initialise pas correctement une constante ou une déclaration de tableau utilise une variable pour spécifier le nombre d'éléments.

ID d'erreur : BC30059

Pour corriger cette erreur

1. Si la déclaration est une instruction **Const**, assurez-vous que la constante est initialisée avec un littéral, une constante précédemment déclarée, un membre d'énumération ou une combinaison de littéraux, constantes et membres d'énumération combinés à des opérateurs.
2. Si la déclaration spécifie un tableau, vérifiez si une variable est utilisée pour spécifier le nombre d'éléments. Si tel est le cas, remplacez la variable par une expression constante.

Voir aussi

Tâches

Comment : déclarer une variable tableau

Référence

Const, instruction (Visual Basic)

L'expression constante n'est pas représentable dans le type '<nomtype>'

Constant expression not representable in type '<typename>'

Vous tentez d'évaluer une constante qui ne correspondra pas au type cible, généralement car elle dépasse la plage.

ID d'erreur : BC30439

Pour corriger cette erreur

- Modifiez le type cible en un type capable de gérer la constante.

Voir aussi

Concepts

Vue d'ensemble des constantes

Autres ressources

Constantes et énumérations (Visual Basic)

Les constantes doivent être de type intrinsèque ou énuméré, et non de type classe, structure, paramètre de type ou tableau

Constants must be of an intrinsic or enumerated type, not a class, structure, type parameter, or array type

Vous avez essayé de déclarer une constante en tant que classe, structure ou type tableau, ou en tant que paramètre de type défini par un type générique conteneur.

Les constantes doivent être de type intrinsèque (**Boolean**, **Byte**, **Date**, **Decimal**, **Double**, **Integer**, **Long**, **Object**, **SByte**, **Short**, **Single**, **String**, **UInteger**, **ULong** ou **UShort**) ou de type **Enum** basé sur l'un des types intégraux.

ID d'erreur : BC30424

Pour corriger cette erreur

1. Déclarez la constante en tant que type intrinsèque ou **Enum**.
2. Une constante peut aussi être une valeur spéciale, telle que **True**, **False** ou **Nothing**. Le compilateur considère que ces valeurs prédéfinies sont du type intrinsèque approprié.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Concepts

Vue d'ensemble des constantes

Types de données en Visual Basic

Autres ressources

Constantes et énumérations (Visual Basic)

Les constantes doivent avoir une valeur

Constants must have a value

Vous avez déclaré une constante sans lui assigner de valeur.

ID d'erreur : BC30438

Pour corriger cette erreur

- Assignez une valeur à la constante.

Voir aussi

Concepts

Vue d'ensemble des constantes

Autres ressources

Constantes et énumérations (Visual Basic)

La contrainte '<contrainte1>' est en conflit avec la contrainte '<contrainte2>' déjà spécifiée pour le paramètre de type '<NomParamètreDeType>'

Constraint '<constraint1>' conflicts with the constraint '<constraint2>' already specified for type parameter '<typeparametername>'

Un type générique est déclaré avec des contraintes incompatibles sur un paramètre de type.

L'instruction suivante peut générer cette erreur.

```
Public Class testClass(Of t As {Structure, Class })
```

Les contraintes `Structure` et `Class` provoquent un conflit pour le paramètre de type `t`, parce que la contrainte **Structure** nécessite que l'argument de type correspondant soit un type valeur, alors que **Class** nécessite qu'il soit un type référence.

ID d'erreur : BC32119

Pour corriger cette erreur

- Modifiez les contraintes de paramètre de type pour éviter des conflits.

Voir aussi

Référence

Liste de types
Structure (Visual Basic)
Class (Visual Basic)

Concepts

Types génériques en Visual Basic
Types valeur et types référence

La contrainte '<contrainte1>' est en conflit avec la contrainte indirecte '<contrainte2>' obtenue de la contrainte de paramètre de type '<ParamètreDeType1>'

Constraint '<constraint1>' conflicts with the indirect constraint '<constraint2>' obtained from the type parameter constraint '<typeparameter1>'

Un type générique est déclaré avec des contraintes incompatibles en raison d'une combinaison de contraintes directes et indirectes.

L'instruction suivante peut générer ce message d'erreur.

```
Public Class testClass(Of t1 As {Structure, t2}, t2 As Class)
```

La contrainte directe `Structure` et la contrainte indirecte `Class` provoquent un conflit pour le paramètre de type `t1`, parce que la contrainte **Structure** requiert que l'argument de type correspondant soit un type valeur, tandis que **Class** requiert qu'il soit un type référence.

ID d'erreur : BC32110

Pour corriger cette erreur

- Modifiez les contraintes du paramètre de type pour éviter des conflits de contraintes.

Voir aussi

Référence

Liste de types
Structure (Visual Basic)
Class (Visual Basic)

Concepts

Types génériques en Visual Basic
Types valeur et types référence

Le type de contrainte '<NomType>' est déjà spécifié pour ce paramètre de type

Constraint type '<typename>' already specified for this type parameter

Une liste de contraintes contient plusieurs fois une contrainte de classe ou d'interface.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Vous pouvez spécifier les éléments requis suivants selon n'importe quelle combinaison :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit hériter d'une classe au plus

Un type ne peut pas implémenter ou hériter plusieurs fois du même type, et vous ne pouvez pas spécifier plusieurs fois un type dans la même liste de contraintes.

ID d'erreur : BC32071

Pour corriger cette erreur

- Supprimez les spécifications redondantes de la même classe ou interface. Elle doit apparaître une seule fois dans la liste des contraintes.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Les contraintes pour ce paramètre de type ne correspondent pas aux contraintes du paramètre de type correspondant défini pour l'un des autres types partiels de '|1'

Constraints for this type parameter do not match the constraints on the corresponding type parameter defined on one of the other partial types of '|1'

Lorsque vous divisez la définition d'une classe ou d'une structure en plusieurs déclarations, le compilateur traite la classe ou la structure comme l'union de toutes ses déclarations partielles. De ce fait, vous ne pouvez pas définir de modificateurs ou de listes de paramètres de type incompatibles dans les différentes déclarations partielles.

ID d'erreur : BC30932

Pour corriger cette erreur

1. Déterminez la liste de paramètres de type voulue pour votre classe ou structure. Vous devez inclure les paramètres, leur ordre et leurs listes de contraintes.
2. Assurez-vous que chaque définition partielle utilise la même liste de paramètres de type.

Voir aussi

Référence

Partial (Visual Basic)

Concepts

Types génériques en Visual Basic

La construction établit une référence indirecte à l'assembly '<NomAssembly>', qui contient '<NomType>'

Message d'erreur
La construction établit une référence indirecte à l'assembly '<NomAssembly>', qui contient <NomType>. Ajoutez une référence de fichier pour <NomFichier> dans votre projet.
Construct makes an indirect reference to assembly '<assemblyname>', which contains <typename>. Add a file reference to <filename> to your project.

Une expression utilise un type, par exemple une classe, une structure, une interface, une énumération ou un délégué, mais votre assembly ne dispose pas de référence de projet à l'assembly qui définit le type.

ID d'erreur : BC31528

Pour corriger cette erreur

- Dans les propriétés de votre projet, ajoutez une référence au fichier contenant l'assembly qui définit le type que vous utilisez.

Voir aussi

Tâches
Comment : modifier des propriétés de projet et des paramètres de configuration
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Résolution d'une référence lorsque plusieurs variables ont le même nom

La construction établit une référence indirecte au projet '<NomProjet>', qui contient '<NomType>'

Message d'erreur
La construction établit une référence indirecte au projet '<NomProjet>', qui contient '<NomType>'. Ajoutez une référence de projet pour '<NomProjet>' dans votre projet.
Construct makes an indirect reference to project '<projectname>', which contains '<typename>'. Add a project reference to '<projectname>' to your project.

Une expression ou une instruction accède à un type défini dans un autre projet, mais votre projet n'a pas de référence directe au projet de définition.

Le type peut être une classe, une structure, une interface, un module ou une énumération.

Le projet qui définit le type cité produit un assembly contenant le type. Si votre projet ne référence pas directement le projet de définition, le compilateur ne peut pas garantir que l'assembly contenant le type se trouve dans la solution, disponible pour l'accès.

ID d'erreur : BC31533

Pour corriger cette erreur

- Déterminez quel projet définit le type cité et ajoutez une référence à ce projet.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Références de projet
Autres ressources
Références aux espaces de noms et aux composants
Gestion des références

Le constructeur '<nom>' ne peut pas s'appeler lui-même

Constructor '<name>' cannot call itself

Une procédure **Sub New** dans une classe ou une structure s'appelle elle-même.

Un constructeur a pour but d'initialiser une instance d'une classe ou d'une structure lorsqu'elle est créée pour la première fois. Une classe ou une structure peut contenir plusieurs constructeurs, à condition qu'ils présentent des listes de paramètres différentes. Un constructeur est autorisé à appeler un autre constructeur pour exécuter ses fonctionnalités en plus des siennes. Mais cela n'a pas de sens qu'un constructeur s'appelle lui-même, et en fait, cela entraînerait une récurrence infinie.

ID d'erreur : BC30298

Pour corriger cette erreur

1. Vérifiez la liste des paramètres du constructeur appelé. Elle doit être différente de celle du constructeur qui effectue l'appel.
2. Si vous ne comptez pas appeler un autre constructeur, supprimez l'intégralité de l'appel **Sub New**.

Voir aussi

Concepts

Durée de vie d'un objet : création et destruction des objets

Un appel à un constructeur est valide uniquement en tant que première instruction dans un constructeur d'instance

Constructor call is valid only as the first statement in an instance constructor

Un appel à **New()** se produit après la première instruction d'un constructeur. Si un constructeur en appelle un autre explicitement, il doit le faire dans la première instruction qui suit l'instruction **Sub New()**.

ID d'erreur : BC30282

Pour corriger cette erreur

- Supprimez l'appel à **New()** ou déplacez-le au début du constructeur.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

Le constructeur doit être déclaré en tant que Sub, et non Function

Constructor must be declared as a Sub, not as a Function

Vous avez tenté de déclarer **Function New**. Les constructeurs doivent être déclarés comme **Sub New**.

ID d'erreur : BC30493

Pour corriger cette erreur

- Utilisez **Sub** au lieu de **Function**.

Voir aussi

Référence

Sub, instruction (Visual Basic)

Les constructeurs ne peuvent pas implémenter des méthodes d'interface

Constructors cannot implement interface methods

Vous avez tenté d'utiliser le mot clé **New** pour implémenter une méthode d'interface, ce qui n'est pas valide.

ID d'erreur : BC30550

Pour corriger cette erreur

- Implémentez les interfaces avec le mot clé **Implements**.

Voir aussi

Référence

Implements, instruction

Concepts

Vue d'ensemble des interfaces

'Continue Do' ne peut apparaître qu'à l'intérieur d'une instruction 'Do'

'Continue Do' can only appear inside a 'Do' statement

Une instruction **Continue Do** peut uniquement apparaître dans une boucle **Do...Loop**.

ID d'erreur : BC30782

Pour corriger cette erreur

1. Si l'instruction **Continue Do** se trouve dans une boucle **For...Next**, remplacez l'instruction par **Continue For**.
2. Si l'instruction **Continue Do** se trouve dans une boucle **While...End While**, remplacez l'instruction par **Continue While**.
3. Supprimez l'instruction **Continue Do** dans le cas contraire.

Voir aussi

Référence

Continue, instruction (Visual Basic)
Do...Loop, instruction (Visual Basic)

'Continue For' ne peut être utilisé qu'à l'intérieur d'une instruction 'For'

'Continue For' can only appear inside a 'For' statement

Une instruction **Continue For** peut uniquement apparaître dans une boucle **For...Next**.

ID d'erreur : BC30783

Pour corriger cette erreur

1. Si l'instruction **Continue For** se trouve dans **Do...Loop**, remplacez l'instruction par **Continue Do**.

   - ou -

   Si l'instruction **Continue For** se trouve dans une boucle **While...End While**, remplacez l'instruction par **Continue While**.

2. Supprimez l'instruction **Continue For** dans le cas contraire.

Voir aussi

Référence

Continue, instruction (Visual Basic)
For...Next, instruction (Visual Basic)

'Continue' doit être suivi de 'Do', 'For' ou 'While'

'Continue' must be followed by 'Do', 'For' or 'While'

Une instruction **Continue** doit être suivie par **Do**, **For** ou **While**, selon que l'instruction **Continue** apparaît dans une boucle **Do...Loop**, une boucle **For...Next** ou une boucle **While...End While**.

ID d'erreur : BC30781

Pour corriger cette erreur

1. Si l'instruction **Continue** se trouve dans une boucle **Do...Loop**, remplacez l'instruction par **Continue Do**.
2. Si l'instruction **Continue** se trouve dans une boucle **For...Next**, remplacez l'instruction par **Continue For**.
3. Si l'instruction **Continue** se trouve dans une boucle **While...End While**, remplacez l'instruction par **Continue While**.
4. Supprimez l'instruction **Continue** dans le cas contraire.

Voir aussi

Référence

Continue, instruction (Visual Basic)

Les instructions 'Continue' ne sont pas valides dans la fenêtre Exécution

'Continue' statements are not valid in the Immediate window

Les instructions **Continue** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30780

Pour corriger cette erreur

- Supprimez l'instruction **Continue** du code de débogage.

Voir aussi

Référence

Continue, instruction (Visual Basic)

Fenêtre Exécution

Autres ressources

Débogage dans Visual Studio

'Continue While' ne peut apparaître que dans une instruction 'While'

'Continue While' can only appear inside a 'While' statement

Une instruction **Continue While** peut uniquement apparaître dans une boucle **For...Next**.

ID d'erreur : BC30784

Pour corriger cette erreur

1. Si l'instruction **Continue While** se trouve dans une boucle **Do...Loop**, remplacez l'instruction par **Continue Do**.
2. Si l'instruction **Continue While** se trouve dans une boucle **For...Next**, remplacez l'instruction par **Continue For**.
3. Supprimez l'instruction **Continue While** dans le cas contraire.

Voir aussi

Référence

Continue, instruction (Visual Basic)
While...End While, instruction (Visual Basic)

Une conversion de '<type1>' en '<type2>' ne peut pas avoir lieu dans une expression constante

Conversion from '<type1>' to '<type2>' cannot occur in a constant expression

L'expression d'initialisation d'une instruction **Const** correspond à un type de données qui ne peut pas être converti dans le type déclaré de la constante.

ID d'erreur : BC30060

Pour corriger cette erreur

- Initialisez la constante avec une expression de type de données pouvant être converti dans le type déclaré pour la constante.

Voir aussi

Référence

Const, instruction (Visual Basic)

Autres ressources

Conversions de type en Visual Basic

Une conversion de '<type1>' en '<type2>' ne peut pas se produire dans une expression constante utilisée en tant qu'argument d'un attribut

Conversion from '<type1>' to '<type2>' cannot occur in a constant expression used as an argument to an attribute

Une expression utilisée pour un argument d'attribut a pour valeur un type de données autre que celui du paramètre d'attribut correspondant, et Visual Basic n'autorise pas la conversion du type requis pour les arguments d'attribut.

Un attribut fournit des métadonnées pour l'élément auquel il s'applique, et le compilateur doit être capable de construire toutes les métadonnées au moment de la compilation. C'est pourquoi, chaque attribut doit utiliser des valeurs constantes au moment de la compilation, et par conséquent, chaque argument d'attribut doit avoir une valeur de constante de compilation.

Certaines conversions de type ne peuvent pas créer des valeurs constantes au moment de la compilation. Par exemple, la conversion de **String** en **Double** ou **Date** dépend des paramètres régionaux au moment de l'exécution. D'autres conversions, par exemple celle d'un tableau d'un type dérivé en un tableau de **Object**, créent des problèmes qui ne permettent pas au compilateur de les autoriser sur les arguments d'attribut.

ID d'erreur : BC30934

Pour corriger cette erreur

- Utilisez une expression qui a pour valeur le même type de données que le paramètre correspondant, telle qu'elle est définie par l'attribut.

Voir aussi

Référence

Const, instruction (Visual Basic)

Concepts

Application des attributs

Autres ressources

Attributs en Visual Basic

Conversions de type en Visual Basic

Pour convertir 'Date' en 'Double', vous devez appeler la méthode 'Date.ToOADate'

Conversion from 'Date' to 'Double' requires calling the 'Date.ToOADate' method

Vous avez tenté d'effectuer un cast d'une valeur **Date** en valeur **Double**, ce qui ne peut pas être effectué sans utiliser la méthode System.DateTime.ToOADate.

ID d'erreur : BC30532

Pour corriger cette erreur

- Utilisez la méthode **System.DateTime.ToOADate** pour convertir la valeur.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

Pour convertir 'Double' en 'Date', vous devez appeler la méthode 'Date.FromOADate'

Conversion from 'Double' to 'Date' requires calling the 'Date.FromOADate'

Vous avez tenté d'effectuer un cast d'une valeur **Date** en valeur **Double**, ce qui ne peut pas être effectué sans utiliser la méthode System.DateTime.FromOADate(System.Double).

ID d'erreur : BC30533

Pour corriger cette erreur

- Utilisez la méthode **FromOADate** pour convertir la valeur.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

Les opérateurs de conversion ne peuvent pas convertir à partir d'un type de base

Conversion operators cannot convert from a base type

Un opérateur de conversion est déclaré avec un type de paramètre d'où dérive le type de retour.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'un type référence vers un type de sa hiérarchie d'héritage, c'est-à-dire un type d'où il dérive ou qui dérive de lui. Une telle conversion peut échouer au moment de l'exécution, mais le compilateur ne peut pas prédire les résultats de l'exécution et permet donc la compilation de cette conversion.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33030

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir à partir d'un type dérivé

Conversion operators cannot convert from a derived type

Un opérateur de conversion est déclaré avec un type de paramètre qui dérive du type de retour.

Au moment de la compilation, Visual Basic étudie l'existence d'une conversion prédéfinie d'un type référence à un type de sa hiérarchie d'héritage, c'est-à-dire un type dont il dérive ou qui en dérive. Une telle conversion peut échouer au moment de l'exécution, mais le compilateur ne peut pas prédire les résultats de l'exécution et permet donc la compilation de cette conversion.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33031

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir d'un type en son type de base

Conversion operators cannot convert from a type to its base type

Un opérateur de conversion est déclaré avec un type de retour d'où dérive le type de paramètre.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'un type référence vers un type de sa hiérarchie d'héritage, c'est-à-dire un type duquel il dérive ou qui dérive de lui. Une telle conversion peut échouer au moment de l'exécution, mais le compilateur ne peut pas prédire les résultats de l'exécution et permet donc la compilation de cette conversion.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33026

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir à partir d'un type en son type dérivé

Conversion operators cannot convert from a type to its derived type

Un opérateur de conversion est déclaré avec un type de retour qui dérive du type de paramètre.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'un type référence vers un type de sa hiérarchie d'héritage, c'est-à-dire un type duquel il dérive ou qui dérive de lui. Une telle conversion peut échouer au moment de l'exécution, mais le compilateur ne peut pas prédire les résultats de l'exécution et permet donc la compilation de cette conversion.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33027

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir d'un type en un type identique

Conversion operators cannot convert from a type to the same type

Un opérateur de conversion est déclaré avec le même type pour le paramètre et le retour.

Cela n'a pas de sens de convertir d'un type en lui-même.

ID d'erreur : BC33024

Pour corriger cette erreur

- Modifiez le type du paramètre ou de la valeur de retour. L'un d'eux doit être du type de la classe ou la structure dans laquelle cet opérateur est défini. L'autre doit être d'un autre type.
- Si vous devez exécuter une transformation sur le contenu du paramètre, utilisez Function, instruction (Visual Basic) pour définir une procédure **Function** au lieu d'un opérateur.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir à partir d'un type interface

Conversion operators cannot convert from an interface type

Un opérateur de conversion est déclaré avec un type interface pour le paramètre.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'une interface à un type référence. Une telle conversion peut échouer au moment de l'exécution, mais le compilateur ne peut pas prédire les résultats de l'exécution et permet donc la compilation de cette conversion.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33029

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir à partir d'Object

Conversion operators cannot convert from Object

Un opérateur de conversion est déclaré avec un paramètre de type de données Object.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'un type référence vers un type de sa hiérarchie d'héritage, c'est-à-dire un type duquel il dérive ou qui dérive de lui. **Object** étant le type de données universel dans le .NET Framework, chaque type dérive de **Object**.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33032

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Object comme type de données universel

Les opérateurs de conversion ne peuvent pas convertir en un type interface

Conversion operators cannot convert to an interface type

Un opérateur de conversion est déclaré avec un type interface comme type de retour.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'un type référence vers une interface. Une telle conversion peut échouer au moment de l'exécution, mais le compilateur ne peut pas prédire les résultats de l'exécution et permet donc la compilation de cette conversion.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33025

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs de conversion ne peuvent pas convertir en Object

Conversion operators cannot convert to Object

Un opérateur de conversion est déclaré avec un type de retour de type de données Object.

Au moment de la compilation, Visual Basic considère qu'une conversion prédéfinie se produit d'un type référence vers un type de sa hiérarchie d'héritage, c'est-à-dire un type duquel il dérive ou qui dérive de lui. **Object** étant le type de données universel dans le .NET Framework, chaque type dérive de **Object**.

Dans la mesure où le compilateur considère cette conversion comme déjà définie, il ne vous permet pas de la redéfinir.

ID d'erreur : BC33028

Pour corriger cette erreur

- Supprimez l'intégralité de cette définition de l'opérateur. Il est déjà prédéfini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Object comme type de données universel

Les opérateurs de conversion doivent être déclarés 'Widening' ou 'Narrowing'

Conversion operators must be declared either 'Widening' or 'Narrowing'

Une instruction Operator ne spécifie ni Widening ni Narrowing.

Lorsque vous définissez un opérateur de conversion, vous devez le déclarer comme **Widening** ou **Narrowing**. Ces caractéristiques étant mutuellement exclusives, vous ne pouvez pas spécifier les deux.

ID d'erreur : BC33017

Pour corriger cette erreur

- Choisissez si l'opérateur de conversion doit être **Widening** ou **Narrowing**, et incluez le mot clé approprié dans l'instruction **Operator**. Vous devez spécifier l'un ou l'autre.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Conversions étendues et restrictives

Procédures d'opérateur

Les conversions de '<nomtype1>' en '<nomtype2>' doivent être explicites

Conversions from '<typename1>' to '<typename2>' must be explicit

Cette conversion ne peut pas être accomplie de manière implicite.

ID d'erreur : BC30664

Pour corriger cette erreur

- Utilisez **CType**, ou l'une des fonctions de conversion spécifiques, telles que **CInt** ou **CDec**.

Voir aussi

Référence

Fonctions de conversion de types de données

La copie de la valeur du paramètre 'ByRef' '<NomParamètre>' dans l'argument correspondant passe du type '<NomType1>' au type '<NomType2>'

Copying the value of 'ByRef' parameter '<parametername>' back to the matching argument narrows from type '<typename1>' to type '<typename2>'

Une procédure est appelée avec un argument qui s'étend au type de paramètre correspondant, et la conversion du paramètre à l'argument est restrictive.

Lorsque vous définissez une classe ou une structure, vous pouvez définir un ou plusieurs opérateurs de conversion pour convertir ce type de classe ou de structure en d'autres types. Vous pouvez également définir des opérateurs de conversion inverse pour convertir ces autres types en votre type de classe ou de structure. Lorsque vous utilisez votre type de classe ou de structure dans un appel de procédure, Visual Basic peut utiliser ces opérateurs de conversion pour convertir le type d'un argument en type de son paramètre correspondant.

Si vous passez l'argument ByRef, Visual Basic copie parfois sa valeur dans une variable locale de la procédure au lieu de passer une référence. Dans ce cas, lors du retour de la procédure, Visual Basic doit copier la valeur de la variable locale dans l'argument du code appelant.

Si une valeur d'argument **ByRef** est copiée dans la procédure et que l'argument et le paramètre sont du même type, aucune conversion n'est nécessaire. Mais si les types sont différents, Visual Basic doit effectuer une conversion dans les deux sens. Si l'un des types est votre type de classe ou de structure, Visual Basic doit le convertir vers et à partir de l'autre type. Si l'une de ces conversions est étendue, la conversion inverse peut être restrictive.

ID d'erreur : BC32053

Pour corriger cette erreur

- Si possible, utilisez un argument d'appel du même type que le paramètre de procédure. Ainsi, Visual Basic n'a pas besoin d'effectuer une conversion.
- Si vous devez appeler la procédure avec un type d'argument autre que le type du paramètre, sans retourner une valeur dans l'argument d'appel, affectez au paramètre la valeur ByVal au lieu de **ByRef**.
- Si vous devez retourner une valeur dans l'argument d'appel, affectez à l'opérateur de conversion inverse la valeur Widening, si possible.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Procédures d'opérateur
Conversions étendues et restrictives

Autres ressources

Conversions de type en Visual Basic

La bibliothèque standard '<NomFichier>' est introuvable.

Could not find standard library: '<filename>'

Visual Basic ne peut pas détecter ou ouvrir l'une des bibliothèques DLL standard requises pour la compilation et la liaison.

La bibliothèque inutilisable doit probablement être mscorlib.dll ou microsoft.visualbasic.dll.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40049

Pour corriger cette erreur

1. Vérifiez que le fichier mentionné dans le message d'erreur est présent sur le disque dur à partir duquel vous exécutez Visual Basic. En général, les bibliothèques standard doivent figurer dans un sous-répertoire sous \WINNT\Microsoft.NET\Framework or \WINDOWS\Microsoft.NET\Framework.
2. Vérifiez que ni le fichier ni le répertoire ne contient un paramètre ou un attribut qui empêche l'accès en lecture par Visual Basic.
3. Vérifiez que le nom de fichier et l'extension sont correctement orthographiés. La mise en majuscules n'a pas d'importance.
4. Si le fichier est correctement enregistré et accessible, il peut être endommagé sur le disque. Réinstallez Visual Basic, si possible.
5. Notez le nom de fichier et l'extension exacts et contactez les services de Support Technique de Microsoft.

Voir aussi

Tâches

Comment : appeler le compilateur de ligne de commande

Concepts

Aide auprès des Services du Support Technique Microsoft (Visual Studio)*

Autres ressources

Génération à partir de la ligne de commande (Visual Basic)

Impossible de trouver la bibliothèque '<nombibliothèque>'

Could not find library '<libraryname>'

La bibliothèque spécifiée n'a pas pu être trouvée.

Ce problème peut se produire lors de la compilation à partir de l'invite de commandes et à l'aide d'un fichier réponse (.rsp) qui référence de nombreux assemblys

ID d'erreur : BC2017

Pour corriger cette erreur

- Vérifiez également que le nom de la bibliothèque est correct.

Voir aussi

Autres ressources

Présentation du débogueur

'D' ne peut plus être utilisé pour indiquer un exposant ; utilisez 'E' à la place

'D' can no longer be used to indicate an exponent, use 'E' instead

Vous ne pouvez pas utiliser le caractère 'D' pour indiquer l'élévation à la puissance.

ID d'erreur : BC30827

Pour corriger cette erreur

- Utilisez l'opérateur **^** ou les caractères **E+** pour indiquer la présence d'un exposant. Par exemple :

```
Const Mole = 6.02E+23 ' Same as 6.02D23
Const Mole2 = 6.02 * 10 ^ 23 ' Same as 6.02D23
```

Voir aussi

Référence

^, opérateur (Visual Basic)

Concepts

Types de données numériques

Le modificateur 'Custom' n'est pas valide pour les événements déclarés sans types délégués explicites

'Custom' modifier is not valid on events declared without explicit delegate types

Contrairement à un événement qui n'est pas personnalisé, une déclaration **Custom Event** requiert une clause **As** après le nom de l'évènement qui spécifie explicitement le type délégué de l'événement.

Les événements non personnalisés peuvent être définis avec une clause **As** et un type délégué explicite, ou avec une liste de paramètres qui suit immédiatement le nom de l'évènement.

ID d'erreur : BC31122

Pour corriger cette erreur

1. Définissez un délégué avec la même liste de paramètres que l'événement personnalisé.

   Par exemple, si **Custom Event** a été défini par `Custom Event Test(ByVal sender As Object, ByVal i As Integer)`, le délégué correspondant est le suivant :

   VB

   ```
   Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)
   ```

2. Remplacez la liste des paramètres de l'événement personnalisé par une clause **As** qui spécifie le type délégué.

   Pour reprendre l'exemple précédent, la déclaration **Custom Event** est réécrite comme suit :

   VB

   ```
   Custom Event Test As TestDelegate
   ```

Exemple

Cet exemple déclare **Custom Event** et spécifie la clause **As** requise avec un type délégué.

VB

```
Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)
Custom Event Test As TestDelegate
    AddHandler(ByVal value As TestDelegate)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As TestDelegate)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal i As Integer)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

Voir aussi

Référence

Custom
Event, instruction
Delegate, instruction

Autres ressources

Événements en Visual Basic

Le modificateur 'Custom' n'est pas valide pour les événements déclarés dans les interfaces

'Custom' modifier is not valid on events declared in interfaces

Un événement personnalisé ne peut pas être déclaré dans une interface parce qu'il doit fournir une implémentation de ses méthodes **AddHandler**, **RemoverHandler** et **RaiseEvent**.

Le mot clé **Custom** peut être utilisé dans une classe dérivée qui implémente l'événement.

ID d'erreur : BC31121

Pour corriger cette erreur

- Supprimez le mot clé **Custom** de la déclaration de l'événement dans l'interface.

Exemple

Cet exemple montre comment implémenter un événement déclaré dans une interface en tant qu'événement personnalisé.

VB

```
Interface TestInterface
    Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)

    Event Test As TestDelegate
End Interface

Class TestClass
    Implements TestInterface

    Public Custom Event Test As TestInterface.TestDelegate _
    Implements TestInterface.Test
        AddHandler(ByVal value As TestInterface.TestDelegate)
            ' Code for adding an event handler goes here.
        End AddHandler

        RemoveHandler(ByVal value As TestInterface.TestDelegate)
            ' Code for removing an event handler goes here.
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal i As Integer)
            ' Code for raising an event goes here.
        End RaiseEvent
    End Event
End Class
```

Voir aussi

Référence

Custom
Event, instruction
Delegate, instruction
Class, instruction (Visual Basic)
Interface, instruction (Visual Basic)

Autres ressources

Événements en Visual Basic

Un modificateur 'Custom' ne peut être utilisé qu'immédiatement avant une déclaration 'Event'

'Custom' modifier can only be used immediately before an 'Event' declaration

Le mot clé **Custom** doit précéder uniquement le mot clé **Event** dans une déclaration d'événement personnalisée.

ID d'erreur : BC31140

Pour corriger cette erreur

- Déplacez le mot clé **Custom** pour qu'il précède le mot clé **Event**.

  - ou -

- Supprimez le mot clé **Custom**.

Voir aussi

Référence

Custom

Event, instruction

Autres ressources

Événements en Visual Basic

'Custom Event' doit se terminer par un 'End Event' correspondant

'Custom Event' must end with a matching 'End Event'

Une déclaration **Custom Event** doit se terminer par une instruction **End Event**

ID d'erreur : BC31114

Pour corriger cette erreur

- Vérifiez que la déclaration **Custom Event** se termine par une instruction **End Event**

Voir aussi

Référence

Event, instruction

'Currency' n'est plus un type pris en charge ; utilisez le type 'Decimal' à la place

'Currency' is no longer a supported type; use the 'Decimal' type instead

Le type de données **Currency** a été remplacé par le type de données **Decimal**.

ID d'erreur : BC30803

Pour corriger cette erreur

- Utilisez le type de données **Decimal** à la place de **Currency**.

Voir aussi

Référence

Decimal, type de données (Visual Basic)

Concepts

Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Les classes dérivées ne peuvent pas déclencher les événements de la classe de base

Derived classes cannot raise base class events

Un événement peut être déclenché uniquement à partir de l'espace de déclaration dans lequel il est déclaré. En conséquence, une classe ne peut pas déclencher d'événements à partir d'une autre classe, même d'une classe dont elle est dérivée.

ID d'erreur : BC30029

Pour corriger cette erreur

- Déplacez l'instruction **Event** ou l'instruction **RaiseEvent** de manière à ce qu'elles soient dans la même classe.

Voir aussi

Tâches

Comment : créer des classes dérivées

Référence

Event, instruction

RaiseEvent, instruction

Les délégués ne peuvent pas implémenter des méthodes d'interface

Delegates cannot implement interface methods

Un délégué est un type référence qui pointe vers une procédure partagée ou une procédure d'instance sur un objet. Compte tenu que la procédure vers laquelle il pointe peut être modifiée par assignation, l'instruction **Delegate** ne peut pas prendre en charge les clauses **Handles** ou **Implements**.

ID d'erreur : BC30018

Pour corriger cette erreur

- Supprimez la clause **Implements** de l'instruction **Delegate**.

Voir aussi

Référence

Delegate, instruction
Handles
Implements, instruction

Concepts

Délégués et opérateur AddressOf

Les délégués ne peuvent pas gérer les événements

Delegates cannot handle events

Un délégué est un type référence qui pointe vers une procédure partagée ou une procédure d'instance sur un objet. Compte tenu que la procédure vers laquelle il pointe peut être modifiée par assignation, l'instruction **Delegate** ne peut pas prendre en charge les clauses **Handles** ou **Implements**.

ID d'erreur : BC30019

Pour corriger cette erreur

- Supprimez la clause **Handles** de l'instruction **Delegate**.

Voir aussi

Référence

Delegate, instruction
Handles
Implements, instruction

Concepts

Délégués et opérateur AddressOf

Le type délégué '<NomDélégué>' de l'événement '<NomÉvénement>' n'est pas conforme CLS

Delegate type '<delegatename>' of event '<eventname>' is not CLS-compliant

Une instruction Event utilise un délégué pour spécifier sa signature, mais une instruction Delegate est marquée comme `<CLSCompliant(False)>` ou n'est pas marquée.

Lorsque vous appliquez l'attribut CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40050

Pour corriger cette erreur

- Si la conformité CLS est requise et si vous avez le contrôle sur la définition du délégué, appliquez **CLSCompliantAttribute** à sa déclaration pour le marquer comme `<CLSCompliant(True)>`.
- Si vous n'avez pas le contrôle sur la définition du délégué ou si vous ne pouvez pas le marquer comme conforme, supprimez **CLSCompliantAttribute** de l'instruction **Event** ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Concepts

Écriture d'un code conforme CLS

Un délégué dans une interface ne peut pas être déclaré '<spécificateur>'

Delegate in an interface cannot be declared '<specifier>'

Un modificateur non valide a été spécifié avec un délégué déclaré dans une interface.

ID d'erreur : BC31068

Pour corriger cette erreur

- Supprimez les modificateurs, tels que **Public**, **Private**, **Shared**, **Friend**, **Protected**, **Protected Friend** et **Overridable**.

Voir aussi

Concepts

Vue d'ensemble des interfaces

Autres ressources

Interfaces en Visual Basic

Le constructeur délégué pour le type '<nomtype>', nécessaire à l'implémentation des délégués, est introuvable

Delegate constructor for type '<typename>', necessary for the implementation of delegates, cannot be found

L'assembly qui contient le type doit être endommagé ou non valide.

ID d'erreur : BC31074

Pour corriger cette erreur

1. Vérifiez que l'assembly qui contient ce type est valide.
2. Recompilez le programme afin de vérifier si l'erreur se reproduit.
3. Si cette erreur persiste, enregistrez votre travail et redémarrez Visual Studio.

Voir aussi

Autres ressources

Support technique et accessibilité

La classe déléguée '<nomclasse>' n'a pas de méthode Invoke, c'est pourquoi une expression de ce type ne peut pas être la cible d'un appel de méthode

Delegate class '<classname>' has no Invoke method, so an expression of this type cannot be the target of a method call

Un appel à **Invoke** via un délégué a échoué car **Invoke** n'est pas implémenté sur la classe déléguée.

ID d'erreur : BC30220

Pour corriger cette erreur

1. Assurez-vous qu'une instance de la classe déléguée a été créée avec une instruction **Dim** et qu'une procédure a été assignée à l'instance déléguée avec l'opérateur **AddressOf**.
2. Repérez le code qui implémente la classe déléguée et vérifiez qu'il implémente la procédure **Invoke**.

Voir aussi

Référence

Delegate, instruction
AddressOf, opérateur
Dim, instruction (Visual Basic)

Autres ressources

Délégués en Visual Basic

Des valeurs par défaut ne peuvent pas être fournies pour des paramètres qui ne sont pas déclarés 'Optional'

Default values cannot be supplied for parameters that are not declared 'Optional'

Les arguments requis doivent être fournis lorsque la procédure est appelée, et ils ne prennent pas les valeurs par défaut dans la déclaration.

ID d'erreur : BC32024

Pour corriger cette erreur

- Déclarez l'argument avec la valeur **Optional** ou supprimez la valeur par défaut.

Voir aussi

Référence

Optional (Visual Basic)

Concepts

Paramètres facultatifs

L'accès à la propriété par défaut est ambigu entre le membre '<nompropriétépardéfaut>' de l'interface héritée '<nominterface1>' et le membre '<nompropriétépardéfaut>' de l'interface héritée '<nominterface2>'

Default property access is ambiguous between the inherited interface members '<defaultpropertyname>' of interface '<interfacename1>' and '<defaultpropertyname>' of interface '<interfacename2>'

Une interface hérite de deux interfaces, chacune déclarant une propriété par défaut avec le même nom. Le compilateur ne peut pas résoudre un accès à cette propriété par défaut sans qualification. L'exemple suivant illustre ce comportement :

```
Public Interface Iface1
    Default Property prop(ByVal arg As Integer) As Integer
End Interface
Public Interface Iface2
    Default Property prop(ByVal arg As Integer) As Integer
End Interface
Public Interface Iface3
    Inherits Iface1, Iface2
End Interface
Public Class testClass
    Public Sub accessDefaultProperty()
        Dim testObj As Iface3
        Dim testInt As Integer = testObj(1)
    End Sub
End Class
```

Lorsque vous spécifiez `testObj(1)`, le compilateur tente de le résoudre en propriété par défaut. Toutefois, puisqu'il existe deux propriétés par défaut possibles en raison des interfaces héritées, le compilateur signale cette erreur.

ID d'erreur : BC30686

Pour corriger cette erreur

- Évitez d'hériter des membres avec le même nom. Dans l'exemple précédent, si `testObj` n'a pas besoin, par exemple, des membres de `Iface2`, déclarez-le comme suit :

  ```
          Dim testObj As Iface1
  ```

  - ou -

- Implémentez l'interface qui hérite dans une classe. Vous pouvez ensuite implémenter chacune des propriétés héritées avec des noms différents. Toutefois, seule l'une d'entre elles peut être la propriété par défaut de la classe d'implémentation. L'exemple suivant illustre ce comportement :

  ```
  Public Class useIface3
      Implements Iface3
      Default Public Property prop1(ByVal arg As Integer) As Integer Implements Iface1.p
  rop
          ' Insert code to define Get and Set procedures for prop1.
      End Property
      Public Property prop2(ByVal arg As Integer) As Integer Implements Iface2.prop
          ' Insert code to define Get and Set procedures for prop2.
      End Property
  End Class
  ```

Voir aussi

Concepts

Vue d'ensemble des interfaces

Propriétés par défaut

La propriété par défaut '<nompropriété1>' est en conflit avec la propriété par défaut '<nompropriété2>' de la classe '<nomclasse>' et devrait donc être déclarée 'Shadows'

Default property '<propertyname1>' conflicts with default property '<propertyname2>' in '<classname>' and so should be declared 'Shadows'

Une propriété est déclarée avec le même nom qu'une propriété définie dans la classe de base. Dans ce cas, la propriété de cette classe doit occulter la propriété de la classe de base.

Ce message est un avertissement. **Shadows** est supposé par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40007

Pour corriger cette erreur

- Ajoutez le mot clé **Shadows** à la déclaration ou changez le nom de la propriété déclarée.

Voir aussi

Référence

Shadows

Concepts

Occultation dans Visual Basic

Le membre '<nommembre>' par défaut n'est pas une propriété

Default member '<membername>' is not a property

Vous avez tenté de déclarer un membre **Default**. Toutefois, le mot clé **Default** est utilisé pour identifier les propriétés par défaut.

ID d'erreur : BC30555

Pour corriger cette erreur

- Vérifiez que vous identifiez une propriété avec **Default** ; sinon, supprimez le mot clé **Default**.

Voir aussi

Référence

Default (Visual Basic)

'Default' ne peut pas être combiné avec '<spécificateur>'

'Default' cannot be combined with '<specifier>'

Vous avez tenté d'associer le mot clé **Default** à un spécificateur, tel que **Shared** ou **Private**, ce qui n'est pas valide dans ce contexte.

ID d'erreur : BC30490

Pour corriger cette erreur

- Supprimez **Default**.

  - ou -

- Supprimez le spécificateur en conflit.

Voir aussi

Référence

Default (Visual Basic)

'Default' ne peut être appliqué qu'à un seul nom de propriété dans un <type>

'Default' can be applied to only one property name in a <type>

Vous avez tenté d'appliquer **Default** à plusieurs noms de propriétés dans une classe, interface ou structure.

ID d'erreur : BC30359

Pour corriger cette erreur

- Conservez une seule déclaration **Default** et supprimez toutes les autres.

Voir aussi

Référence

Default (Visual Basic)

Les instructions 'Declare' dans une structure ne peuvent pas être déclarées '<spécificateur>'

'Declare' statements in a structure cannot be declared '<specifier>'

Une déclaration **Declare** possède un spécificateur qui n'est pas valide dans une déclaration **Structure**.

ID d'erreur : BC30791

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Delegate, instruction
Structure, instruction

Les instructions 'Declare' dans un Module ne peuvent pas être déclarées '<spécificateur>'

Declare statements in a Module cannot be declared '<specifier>'

Une déclaration **Declare** possède un spécificateur qui n'est pas valide dans une déclaration **Module**. Les modules ne peuvent jamais être instanciés, ne prennent pas en charge l'héritage et ne peuvent pas implémenter des interfaces.

ID d'erreur : BC30786

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Delegate, instruction
Module, instruction

Les instructions 'Declare' ne sont pas autorisées dans les types génériques ou les types contenus dans des types génériques

'Declare' statements are not allowed in generic types or types contained in generic types

Une instruction **Declare** apparaît dans le cadre d'une classe ou d'une structure générique, ou encore dans le cadre d'une classe ou d'une structure déclarée dans une classe ou une structure générique.

À l'heure actuelle, Visual Basic et .NET Framework ne prennent en charge aucune combinaison de références externes et de types génériques. Pour pouvoir l'appeler correctement, le compilateur nécessite tous les paramètres et le type de retour d'une procédure externe.

ID d'erreur : BC32075

Pour corriger cette erreur

- Déplacez l'instruction **Declare** hors du type générique ou supprimez-le entièrement.

Voir aussi

Référence

Declare, instruction

Concepts

Types génériques en Visual Basic

Déclaration attendue

Declaration expected

Une instruction non déclarative, telle qu'une assignation ou une instruction de boucle, apparaît à l'extérieur de toute procédure. Seules les déclarations sont autorisées à l'extérieur des procédures.

En outre, un élément de programmation est déclaré sans mot clé de déclaration, tel que **Dim** ou **Const**.

ID d'erreur : BC30188

Pour corriger cette erreur

- Déplacez l'instruction non déclarative vers le corps d'une procédure.
- Commencez la déclaration par un mot clé de déclaration approprié.
- Vérifiez si un mot clé de déclaration est mal orthographié.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Procédures dans Visual Basic

Constante de date non valide

Date constant is not valid

Une constante de date a été déclarée avec une date non valide.

ID d'erreur : BC31085

Pour corriger cette erreur

- Corrigez la date utilisée dans la constante.

Voir aussi

Référence

Type de données date (Visual Basic)

Autres ressources

Constantes et énumérations (Visual Basic)

'Do' doit se terminer par un 'Loop' correspondant

'Do' must end with a matching 'Loop'

Une instruction **Do** apparaît sans instruction **Loop** correspondante. Une instruction **Loop** doit être utilisée pour terminer la boucle **Do**.

ID d'erreur : BC30083

Pour corriger cette erreur

- Si cette boucle **Do** fait partie d'un ensemble de boucles imbriquées, vérifiez que chaque boucle est correctement terminée.
- Ajoutez une instruction **Loop** à la fin de la boucle **Do**.

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)

Une division par zéro a eu lieu lors de l'évaluation de cette expression

Division by zero occurred while evaluating this expression

Lors de l'évaluation de cette expression, vous avez essayé de diviser un nombre par zéro.

ID d'erreur : BC30542

Pour corriger cette erreur

- Vérifiez vos valeurs de constantes et de variables pour déterminer comment zéro se produit dans le dénominateur de votre expression.

Voir aussi

Référence

Opérateurs d'assignation

'ElseIf' doit être précédé d'un 'If' ou 'ElseIf' correspondant

'ElseIf' must be preceded by a matching 'If' or 'ElseIf'

Une instruction **ElseIf** apparaît sans une instruction **If** correspondante. Une instruction **ElseIf** doit être précédée d'une instruction **If** ou d'une autre instruction **ElseIf**.

ID d'erreur : BC36005

Pour corriger cette erreur

1. Si ce bloc **If** fait partie d'un ensemble de structures de contrôle imbriquées, vérifiez que chaque structure est correctement terminée.
2. Vérifiez que les autres structures de contrôle imbriquées dans ce bloc **If** sont correctement terminées.
3. Vérifiez que ce bloc **If** est correctement mis en forme.

Voir aussi

Référence

If...Then...Else, instruction (Visual Basic)

Concepts

Structures de décision

'Else' doit être précédé d'un 'If' ou 'ElseIf' correspondant

'Else' must be preceded by a matching 'If' or 'ElseIf'

Une instruction **Else** apparaît sans une instruction **If** correspondante. Une instruction **Else** doit être précédée d'une instruction **If**.

ID d'erreur : BC30086

Pour corriger cette erreur

1. Si ce bloc **If** fait partie d'un ensemble de blocs **If** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **If** sont correctement terminées.
3. Vérifiez que ce bloc **If** est correctement mis en forme.

Voir aussi

Référence

If...Then...Else, instruction (Visual Basic)

Le type de paramètre ou le type de retour de cet opérateur de conversion doit être du type conteneur

Either the parameter type or the return type of this conversion operator must be the containing type

Une définition d'un opérateur de conversion spécifie le paramètre et le type de retour avec des types autres que ceux de la classe ou de la structure dans laquelle l'opérateur est défini.

Lorsque vous définissez un opérateur de conversion dans une classe ou une structure, il doit convertir vers ou à partir du type de cette classe ou structure.

ID d'erreur : BC33022

Pour corriger cette erreur

- Remplacez le type de paramètre ou le type de retour par le type de la classe ou de la structure dans laquelle l'opérateur est défini.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Erreur lors de la création de ressources Win32 : <message d'erreur>

Error creating Win32 resources: <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la création d'une ressource en mémoire. Il peut s'agir d'un problème avec l'environnement ou bien votre ordinateur peut manquer de mémoire.

ID d'erreur : BC30136

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Erreur lors de la création du manifeste d'assembly : <message d'erreur>

Error creating assembly manifest: <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de l'étape de pré-émission de création de l'assembly.

Cela peut se produire en cas de problème avec le fichier de clé ou le conteneur de clé spécifié. Pour signer complètement un assembly, vous devez fournir un fichier de clé valide qui contient des informations sur les clés publique et privée. Pour temporiser la signature d'un assembly, vous devez activer la case à cocher **Temporiser la signature uniquement** et fournir un fichier de clé valide qui contient des informations sur la clé publique. La clé privée n'est pas nécessaire lorsqu'un assembly est à signature différée. Pour plus d'informations, consultez Comment : signer un assembly (Visual Studio).

ID d'erreur : BC30140

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils au sujet de l'erreur AL1019
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Tâches

Comment : signer un assembly (Visual Studio)

Référence

Page Signature, Concepteur de projets
Assembly Linker (Al.exe)
Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Les enums doivent être déclarés en tant que type intégral

Enums must be declared as an integral type

Les seuls types valides que vous pouvez utiliser dans des énumérations sont **SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** et **ULong**. Aucun autre type de données ne peut être utilisé.

ID d'erreur : BC30650

Pour corriger cette erreur

- Spécifiez un type de données de **SByte**, **Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long** ou **ULong**.

Voir aussi

Référence

Liste des types de données (Visual Basic)
Enum, instruction (Visual Basic)

'Enum' doit se terminer par un 'End Enum' correspondant

'Enum' must end with a matching 'End Enum'

Une instruction **Enum** apparaît sans instruction **End Enum** correspondante. Une instruction **End Enum** doit être utilisée pour terminer l'énumération.

ID d'erreur : BC30185

Pour corriger cette erreur

1. Vérifiez que les membres **Enum** sont correctement mis en forme.
2. Ajoutez une instruction **End Enum** à la fin de l'énumération.

Voir aussi

Référence

Enum, instruction (Visual Basic)

Un enum dans une interface ne peut pas être déclaré '<spécificateur>'

Enum in an interface cannot be declared '<specifier>'

Un modificateur non valide a été spécifié avec une énumération déclarée dans une interface.

ID d'erreur : BC31069

Pour corriger cette erreur

- Supprimez les modificateurs, tels que **Public**, **Private**, **Shared**, **Friend**, **Protected**, **Protected Friend** ou **Overridable**.

Voir aussi

Référence

Enum, instruction (Visual Basic)

Autres ressources

Interfaces en Visual Basic

Enum '<nomenum>' doit contenir au moins un membre

Enum '<enumname>' must contain at least one member

Une énumération vide apparaît. Au moins un membre doit être déclaré entre l'instruction **Enum** et l'instruction **End Enum**

ID d'erreur : BC30280

Pour corriger cette erreur

- Ajoutez une déclaration de membre à l'énumération.

Voir aussi

Référence

Enum, instruction (Visual Basic)

Les instructions 'EndIf' ne sont plus prises en charge ; utilisez 'End If' à la place

'EndIf' statements are no longer supported; use 'End If' instead

L'instruction **EndIf** n'est plus prise en charge.

ID d'erreur : BC30826

Pour corriger cette erreur

- Utilisez **End If** pour terminer les instructions **If**.

Voir aussi

Référence

If...Then...Else, instruction (Visual Basic)

'End With' doit être précédé d'un 'With' correspondant

'End With' must be preceded by a matching 'With'

Une instruction **End With** apparaît sans une instruction **With** correspondante. Une instruction **End With** doit être précédée d'une instruction **With** correspondante.

ID d'erreur : BC30093

Pour corriger cette erreur

1. Si ce bloc **With** fait partie d'un ensemble de blocs **With** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **With** sont correctement terminées.
3. Vérifiez que ce bloc **With** est correctement mis en forme.

Voir aussi

Référence

With...End With, instruction (Visual Basic)

'End While' doit être précédé d'un 'While' correspondant

'End While' must be preceded by a matching 'While'

Une instruction **End While** apparaît sans une instruction **While** correspondante. Une instruction **End While** doit être précédée d'une instruction **While** correspondante.

ID d'erreur : BC30090

Pour corriger cette erreur

1. Si ce bloc **While** fait partie d'un ensemble de blocs **While** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **While** sont correctement terminées.
3. Vérifiez que ce bloc **While** est correctement mis en forme.

Voir aussi

Référence

While...End While, instruction (Visual Basic)

'End Using' doit être précédé d'un 'Using' correspondant

'End Using' must be preceded by a matching 'Using'

Une instruction **End Using** apparaît et aucune déclaration **Using** correspondante ne la précède.

ID d'erreur : BC36007

Pour corriger cette erreur

- Supprimez l'instruction **End Using** si elle est redondante.
- Fournissez la procédure Using, instruction (Visual Basic) si elle est manquante.
- Déplacez l'instruction **End Using** vers l'emplacement approprié dans votre code.

Voir aussi

Référence

End (Visual Basic)

'End Try' doit être précédé d'un 'Try' correspondant

'End Try' must be preceded by a matching 'Try'

End Try est utilisé pour terminer un bloc **Try** ; elle ne peut donc apparaître qu'à la fin du bloc. Vous disposez d'une instruction **End Try** redondante ou votre instruction **End Try** apparaît en dehors des limites de son bloc **Try** correspondant.

ID d'erreur : BC30383

Pour corriger cette erreur

1. Recherchez et supprimez l'instruction **End Try** inutile.
2. Déplacez l'instruction **End Try** vers l'emplacement approprié dans votre code.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'End SyncLock' doit être précédé d'un 'SyncLock' correspondant

'End SyncLock' must be preceded by a matching 'SyncLock'

Les blocs **SyncLock** commencent par le mot clé **SyncLock** et se terminent par la construction **End SyncLock**.

ID d'erreur : BC30674

Pour corriger cette erreur

- Vérifiez que le bloc **SyncLock** commence par une instruction **SyncLock**.

Voir aussi

Référence

SyncLock, instruction

'End Sub' doit être la première instruction d'une ligne

'End Sub' must be the first statement on a line

Une instruction **End Sub** suit un séparateur d'instruction deux-points (:). **End Sub** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC32031

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Sub, instruction (Visual Basic)

'End Sub' doit être précédée d'une instruction 'Sub' correspondante

'End Sub' must be preceded by a matching 'Sub'

Une instruction **End Sub** apparaît dans votre code et aucune définition de procédure **Sub** correspondante ne la précède.

ID d'erreur : BC30429

Pour corriger cette erreur

- Supprimez l'instruction **End Sub** si elle est redondante.
- Fournissez la procédure **Sub** si elle est manquante.
- Déplacez **End Sub** vers l'emplacement approprié dans le code.

Voir aussi

Référence

End (Visual Basic)

Concepts

Sub, procédures

'End Sub' attendu

'End Sub' expected

Une instruction **End** doit correspondre au bloc qu'elle termine. Une définition de procédure **Sub** n'a pas été terminée correctement.

ID d'erreur : BC30026

Pour corriger cette erreur

- Ajoutez une instruction **End Sub** à la fin de la procédure.

Voir aussi

Référence

End, instruction

Concepts

Sub, procédures

'End Structure' doit être précédé d'un 'Structure' correspondant

'End Structure' must be preceded by a matching 'Structure'

L'instruction **End Structure** ne peut être utilisée qu'à la fin d'un bloc de structure commençant par une instruction **Structure**.

ID d'erreur : BC30621

Pour corriger cette erreur

- Vérifiez que le bloc de structure en cours commence par une instruction **Structure**.

Voir aussi

Concepts

Structures et classes

Autres ressources

Structures constituées à partir de vos propres types de données

Les instructions 'End' ne sont pas valides dans la fenêtre Exécution

'End' statements are not valid in the Immediate window

Les instructions **Stop** et **End** interrompent l'exécution et ne sont pas autorisées dans un contexte de débogage.

ID d'erreur : BC30123

Pour corriger cette erreur

- N'émettez pas d'instruction **End** ou **Stop** dans la fenêtre **Exécution**.

Voir aussi

Référence

Fenêtre Exécution

Instruction 'End' non valide

'End' statement not valid

Le mot clé **End** est utilisé de manière incorrecte ou est inapproprié dans le contexte en cours.

ID d'erreur : BC30678

Pour corriger cette erreur

1. Utilisez **End** pour qu'une procédure perde sa portée.
2. Utilisez **End** suivi d'autres mots clés pour terminer un bloc de code tel qu'une structure ou une propriété.

Voir aussi

Référence

End, instruction
End (Visual Basic)

L'instruction 'End' ne peut pas être utilisée dans les projets Bibliothèques de classes

'End' statement cannot be used in class library projects

Les projets Bibliothèques de classes utilisés pour créer les DLL ne permettent pas l'utilisation du mot clé **End** pour arrêter l'exécution du code dans une procédure.

ID d'erreur : BC30615

Pour corriger cette erreur

- Utilisez des structures de contrôle, telles que **While** et **For** pour contrôler le déroulement de l'exécution du programme.

Voir aussi

Autres ressources

Flux de contrôle en Visual Basic

'End Set' doit être la première instruction d'une ligne

'End Set' must be the first statement on a line

Une instruction **End Set** suit un séparateur d'instruction deux-points (:). **End Set** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC32034

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Set, instruction (Visual Basic)

'End Set' doit être précédé d'un 'Set' correspondant

'End Set' must be preceded by a matching 'Set'

End Set est utilisé pour terminer les procédures de propriété **Set**. La construction **End Set** a été rencontrée à l'extérieur d'une procédure de propriété **Set**.

ID d'erreur : BC30632

Pour corriger cette erreur

1. Vérifiez que la procédure de propriété **Set** est déclarée après un mot clé **Property** et avant la construction **End Property**.
2. Vérifiez que la procédure de propriété **Set** commence par le mot clé **Set** et se termine par une construction **End Set**.

Voir aussi

Référence

Property, instruction

Concepts

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

'End Select' doit être précédé d'un 'Select Case' correspondant

'End Select' must be preceded by a matching 'Select Case'

Une instruction **End Select** apparaît sans instruction **Select** ou **Select Case** correspondante. Une instruction **End Select** doit être précédée d'une instruction **Select** ou **Select Case**.

ID d'erreur : BC30088

Pour corriger cette erreur

1. Si ce bloc **Select** fait partie d'un ensemble de blocs **Select** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **Select** sont correctement terminées.
3. Vérifiez que ce bloc **Select** est correctement mis en forme.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

'End RemoveHandler' doit être la première instruction sur une ligne

'End RemoveHandler' must be the first statement on a line

Une instruction **End RemoveHandler** suit un séparateur d'instruction deux-points (:). **End RemoveHandler** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC31119

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

RemoveHandler, instruction

Event, instruction

'End RemoveHandler' doit être précédé d'une déclaration 'RemoveHandler' correspondante

'End RemoveHandler' must be preceded by a matching 'RemoveHandler' declaration

Une instruction **End RemoveHandler** est apparue sans une instruction **RemoveHandler** correspondante. L'instruction **End RemoveHandler** doit être précédée d'une instruction **RemoveHandler** correspondante.

ID d'erreur : BC31125

Pour corriger cette erreur

- Vérifiez que l'instruction **RemoveHandler** précédente est valide et correctement orthographiée.

Voir aussi

Référence

RemoveHandler, instruction
Event, instruction

'End RaiseEvent' doit être la première instruction sur une ligne

'End RaiseEvent' must be the first statement on a line

Une instruction **End RaiseEvent** suit un séparateur d'instruction deux-points (:). **End RaiseEvent** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC31120

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

RaiseEvent, instruction

Event, instruction

'End RaiseEvent' doit être précédé d'une déclaration 'RaiseEvent' correspondante

'End RaiseEvent' must be preceded by a matching 'RaiseEvent' declaration

Une instruction **End RaiseEvent** est apparue sans une instruction **RaiseEvent** correspondante. L'instruction **End RaiseEvent** doit être précédée d'une instruction **RaiseEvent** correspondante.

ID d'erreur : BC31126

Pour corriger cette erreur

- Vérifiez que l'instruction **RaiseEvent** précédente est valide et correctement orthographiée.

Voir aussi

Référence

RaiseEvent, instruction
Event, instruction

'End Property' doit être précédée d'une 'Property' correspondante

'End Property' must be preceded by a matching 'Property'

Une instruction **End Property** apparaît dans votre code et aucune déclaration **Property** correspondante ne la précède.

ID d'erreur : BC30431

Pour corriger cette erreur

- Supprimez l'instruction **End Property** si elle est redondante.
- Fournissez la procédure **Property** si elle est manquante.
- Déplacez **End Property** vers l'emplacement approprié dans le code.

Voir aussi

Référence

End (Visual Basic)

Concepts

Propriétés et procédures Property

'End Operator' doit être la première instruction d'une ligne

'End Operator' must be the first statement on a line

Une instruction **End Operator** suit un séparateur d'instruction deux-points (:). **End Operator** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC33006

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code
Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

Operator, instruction

'End Operator' doit être précédé d'un 'Operator' correspondant

'End Operator' must be preceded by a matching 'Operator'

Une instruction **End Operator** apparaît et aucune déclaration **Operator** correspondante ne la précède.

ID d'erreur : BC33007

Pour corriger cette erreur

- Supprimez l'instruction **End Operator** si elle est redondante.
- Fournissez la procédure **Operator** si elle est manquante.
- Déplacez l'instruction **End Operator** vers l'emplacement approprié dans votre code.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

End (Visual Basic)

Operator, instruction

Concepts

Procédures d'opérateur

'End Operator' attendu

'End Operator' expected

Une instruction **End** doit correspondre au bloc qu'elle termine. Une définition de procédure **Operator** n'a pas été terminée correctement.

ID d'erreur : BC33005

Pour corriger cette erreur

- Ajoutez une instruction **End Operator** à la fin de la procédure.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

End, instruction

Concepts

Procédures d'opérateur

Fin d'instruction attendue

End of statement expected

L'instruction est complète sur le plan de la syntaxe, mais un élément de programmation supplémentaire suit l'élément qui termine l'instruction. Un terminateur de ligne est requis à la fin de chaque instruction.

ID d'erreur : BC30205

Pour corriger cette erreur

1. Vérifiez si deux instructions différentes ont été placées par inadvertance sur la même ligne.
2. Insérez un terminateur de ligne après l'élément qui termine l'instruction.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Concepts

Vue d'ensemble des instructions

Fin de liste de paramètres attendue

End of parameter list expected

Une spécification **ParamArray** est suivie d'une ou plusieurs autres spécifications d'arguments dans une déclaration de procédure.

ID d'erreur : BC30192

Pour corriger cette erreur

- Déplacez la spécification **ParamArray** à la fin de la liste des arguments.

Voir aussi

Concepts

Tableaux de paramètres

Fin d'expression attendue

End of expression expected

Des caractères supplémentaires ont été trouvés à la fin d'une expression par ailleurs complète.

ID d'erreur : BC30710

Pour corriger cette erreur

- Vérifiez la syntaxe de l'objet ou de l'instruction.

Voir aussi

Autres ressources

Débogage dans Visual Studio

'End Namespace' doit être précédé d'un 'Namespace' correspondant

'End Namespace' must be preceded by a matching 'Namespace'

Les blocs namespace commencent par une instruction **Namespace** et se terminent par une construction **End Namespace**.

ID d'erreur : BC30623

Pour corriger cette erreur

- Vérifiez que le bloc namespace commence par une instruction **Namespace**.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

'End Module' doit être précédé d'un 'Module' correspondant

'End Module' must be preceded by a matching 'Module'

Les blocs module commencent par une instruction **Module** et se terminent par une construction **End Module**.

ID d'erreur : BC30622

Pour corriger cette erreur

- Vérifiez que le bloc module commence par une instruction **Module**.

Voir aussi

Référence

Module, instruction

'End Interface' doit être précédé d'un 'Interface' correspondant

'End Interface' must be preceded by a matching 'Interface'

Une instruction **End Interface** apparaît sans une instruction **Interface** correspondante. Une instruction **End Interface** doit être précédée d'une instruction **Interface** correspondante.

ID d'erreur : BC30252

Pour corriger cette erreur

1. Vérifiez si une instruction **Interface** précédente n'est pas mal orthographiée ou non valide.
2. Vérifiez que les membres **Interface** sont correctement mis en forme.

Voir aussi

Référence

Interface, instruction (Visual Basic)

'End If' doit être précédé d'un 'If' correspondant

'End If' must be preceded by a matching 'If'

Une instruction **End If** apparaît sans une instruction **If** correspondante. Une instruction **End If** doit être précédée d'une instruction **If**.

ID d'erreur : BC30087

Pour corriger cette erreur

1. Si ce bloc **If** fait partie d'un ensemble de blocs **If** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **If** sont correctement terminées.
3. Vérifiez que ce bloc **If** est correctement mis en forme.

Voir aussi

Référence

If...Then...Else, instruction (Visual Basic)

'End Get' doit être la première instruction d'une ligne

'End Get' must be the first statement on a line

Une instruction **End Get** suit un séparateur d'instruction deux-points (:). **End Get** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC32033

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Get, instruction

'End Get' doit être précédé d'un 'Get' correspondant

'End Get' must be preceded by a matching 'Get'

End Get est utilisé pour terminer les procédures de propriété **Get**. La construction **End Get** a été rencontrée à l'extérieur d'une procédure de propriété **Get**.

ID d'erreur : BC30630

Pour corriger cette erreur

1. Vérifiez que la procédure de propriété **Get** est déclarée après un mot clé **Property** et avant la construction **End Property**.
2. Vérifiez que la procédure de propriété **Get** commence par le mot clé **Get** et se termine par une construction **End Get**.

Voir aussi

Référence

Property, instruction

Concepts

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

'End Function' doit être la première instruction d'une ligne

'End Function' must be the first statement on a line

Une instruction **End Function** suit un séparateur d'instruction deux-points (:). **End Function** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC32032

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Function, instruction (Visual Basic)

'End Function' doit être précédée d'une 'Function' correspondante

'End Function' must be preceded by a matching 'Function'

Une instruction **End Function** apparaît dans votre code et aucune définition de procédure **Function** correspondante ne la précède.

ID d'erreur : BC30430

Pour corriger cette erreur

1. Supprimez l'instruction **End Function** si elle est redondante.
2. Fournissez la procédure **Function** si elle est manquante.
3. Déplacez **End Function** vers l'emplacement approprié dans le code.

Voir aussi

Référence

End (Visual Basic)

Concepts

Function, procédures

'End Function' attendu

'End Function' expected

Une instruction **End** doit correspondre au bloc qu'elle termine. Une définition de procédure **Function** n'a pas été terminée correctement.

ID d'erreur : BC30027

Pour corriger cette erreur

- Ajoutez une instruction **End Function** à la fin de la procédure.

Voir aussi

Référence

End, instruction

Concepts

Function, procédures

'End Event' doit être précédé d'un 'Custom Event' correspondant

'End Event' must be preceded by a matching 'Custom Event'

Une instruction **End Event** apparaît sans une instruction **Custom Event** correspondante. Une instruction **End Event** doit être précédée d'une instruction **Custom Event** correspondante.

ID d'erreur : BC31123

Pour corriger cette erreur

- Vérifiez qu'une instruction **Custom Event** précédente est valide et correctement orthographiée.

Voir aussi

Référence

Event, instruction

'End Enum' doit être précédé d'un 'Enum' correspondant

'End Enum' must be preceded by a matching 'Enum'

Une instruction **End Enum** apparaît sans une instruction **Enum** correspondante. Une instruction **End Enum** doit être précédée d'une instruction **Enum** correspondante.

ID d'erreur : BC30184

Pour corriger cette erreur

1. Assurez-vous qu'une instruction **Enum** précédente n'est pas mal orthographiée ou non valide.
2. Vérifiez que les membres **Enum** sont correctement mis en forme.

Voir aussi

Référence

Enum, instruction (Visual Basic)

'End Class' doit être précédée d'une instruction 'Class' correspondante

'End Class' must be preceded by a matching 'Class'

End Class est utilisée pour terminer un bloc **Class** ; elle ne peut donc apparaître qu'à la fin du bloc. Vous disposez d'une instruction **End Class** redondante ou votre instruction **End Class** apparaît en dehors des limites de son bloc **Class** correspondant.

ID d'erreur : BC30460

Pour corriger cette erreur

- Recherchez et supprimez les instructions **End Class** redondantes.
- Déplacez l'instruction **End Class** vers l'emplacement approprié dans votre code.

Voir aussi

Référence

End (Visual Basic)

'End AddHandler' doit être la première instruction sur une ligne

'End AddHandler' must be the first statement on a line

Une instruction **End AddHandler** suit un séparateur d'instruction deux-points (:). **End AddHandler** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC31118

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

AddHandler, instruction

Event, instruction

'End AddHandler' doit être précédé d'une déclaration 'AddHandler' correspondante

'End AddHandler' must be preceded by a matching 'AddHandler' declaration

Une instruction **End AddHandler** est apparue sans une instruction **AddHandler** correspondante. L'instruction **End AddHandler** doit être précédée d'une instruction **AddHandler** correspondante.

ID d'erreur : BC31124

Pour corriger cette erreur

- Vérifiez qu'une instruction **AddHandler** précédente est valide et correctement orthographiée.

Voir aussi

Référence

AddHandler, instruction
Event, instruction

Erreur dans l'import au niveau du projet '<NomÉlémentQualifié>' à '<NomConteneurQualifié>' : <MessageErreur>

Error in project-level import '<qualifiedelementname>' at '<qualifiedcontainername>' : <errormessage>

Une instruction accède à un élément de programmation qui est défini dans un autre assembly, mais il n'existe aucune référence de projet à cet assembly.

Par exemple, votre code peut accéder à une énumération nommée `desiredEnumeration` à l'aide de la chaîne de qualification `otherNamespace.otherClass.desiredEnumeration`. Si le compilateur ne parvient pas à trouver `otherNamespace.otherClass` parmi les références de votre projet, il génère cette erreur.

ID d'erreur : BC30797

Pour corriger cette erreur

1. Assurez-vous que chaque élément de la chaîne de qualification de l'élément de programmation est correctement orthographié.
2. Assurez-vous que votre projet contient une référence à l'assembly qui définit l'élément de programmation souhaité.
3. Consultez le message d'erreur et prenez les mesures qui s'imposent.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration
Comment : ajouter ou supprimer des références en Visual Studio

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

Erreur lors de l'enregistrement du fichier de ressources Win32 temporaire '<nomfichier>' : <message d'erreur>

Error saving temporary Win32 resource file '<filename>': <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de l'obtention d'un nom de fichier à utiliser pour écrire une ressource en mémoire.

ID d'erreur : BC30137

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Erreur lors de la définition de l'option du manifeste d'assembly : <message d'erreur>

Error setting assembly manifest option: <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la recherche d'un fichier de clé ou d'un nom de clé avec lequel signer l'assembly.

ID d'erreur : BC30139

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la page intitulée Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Les instructions 'Error' ne sont pas valides dans la fenêtre Exécution

'Error' statements are not valid in the Immediate window

Les instructions **Error** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30724

Pour corriger cette erreur

- Supprimez les instructions **Error** du code de débogage.

Voir aussi

Référence

Error, instruction

Autres ressources

Débogage dans Visual Studio

Arrêt de l'évaluation de l'expression ou de l'instruction

Evaluation of expression or statement stopped

L'élément de menu **Suspendre l'évaluation** a été sélectionné, provoquant ainsi l'arrêt de l'évaluation.

ID d'erreur : BC30721

Pour corriger cette erreur

- Désélectionnez **Suspendre l'évaluation** pour que l'exécution se poursuive.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Délai d'attente dépassé pour l'évaluation de l'expression ou de l'instruction

Evaluation of expression or statement timed out

L'évaluation d'une expression ne s'est pas terminée à temps.

ID d'erreur : BC30722

Pour corriger cette erreur

1. Vérifiez que le code entré est correct.
2. Simplifiez votre expression afin de réduire son temps d'exécution.

Voir aussi

Autres ressources

Débogage dans Visual Studio

L'événement '<nomévénement>' est introuvable

Event '<eventname>' cannot be found

Vous avez tenté de gérer un événement que le compilateur ne peut pas localiser.

ID d'erreur : BC30590

Pour corriger cette erreur

- Assurez-vous que le nom de l'événement est correctement orthographié.

Voir aussi

Concepts

Événements et gestionnaires d'événements

L'événement '<nomévénement>' ne peut pas être géré car il n'est pas accessible à partir de '<nom>'

Event '<eventname>' cannot be handled because it is not accessible from '<name>'

Vous avez tenté de gérer un événement qui n'est pas accessible. Par exemple, si une variable **Handles** est partagée, la méthode gérant les événements doit également être partagée.

ID d'erreur : BC30585

Pour corriger cette erreur

- Assurez-vous que l'événement est accessible.

Voir aussi

Concepts

Événements et gestionnaires d'événements

L'événement '<nomévénement>' spécifié par l'attribut 'DefaultEvent' n'est pas un événement accessible publiquement pour cette classe

Event '<eventname>' event specified by the 'DefaultEvent' attribute is not a publicly accessible event for this class

L'attribut DefaultEventAttribute doit spécifier le nom d'un événement accessible publiquement dans la classe à laquelle l'attribut est appliqué.

ID d'erreur : BC30770

Pour corriger cette erreur

1. Assurez-vous que la classe définit un événement accessible publiquement.
2. Assurez-vous que le nom de l'événement dans la classe correspond au nom spécifié par l'attribut **DefaultEventAttribute**.

Voir aussi

Référence

Event, instruction
Class, instruction (Visual Basic)
Public (Visual Basic)
DefaultEventAttribute

L'événement '<nomévénement>' déclare implicitement '<nommembre>', qui est en conflit avec un membre dans le <type> '<nomclasse>' de base, et devrait donc être déclaré 'Shadows'

Event '<eventname>' implicitly declares '<membername>', which conflicts with a member in the base <type> '<classname>', and so the event should be declared 'Shadows'

Un événement est déclaré avec un nom qui est utilisé pour former un membre implicite portant le même nom qu'un membre de la classe de base. Par exemple, si vous déclarez un événement nommé **Event1**, le compilateur génère les procédures implicites **add_Event1** et **remove_Event1**. Si la classe de base comporte un membre portant l'un de ces noms, l'événement de cette classe doit occulter le membre de la classe de base.

Ce message est un avertissement. **Shadows** est supposé par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40012

Pour corriger cette erreur

1. Pour masquer le membre de la classe de base, ajoutez le mot clé **Shadows** à la déclaration d'événement.
2. Dans le cas contraire, changez le nom de l'événement.

Voir aussi

Référence

Event, instruction

Shadows

Concepts

Occultation dans Visual Basic

L'événement '<nomévénement1>' ne peut pas implémenter l'événement '<nomévénement2>', car son type délégué diffère du type délégué d'un autre événement implémenté par '<nomévénement1>'

Event '<eventname1>' cannot implement event '<eventname2>' because its delegate type does not match the delegate type of another event implemented by '<eventname1>'

Visual Basic ne peut pas implémenter d'événement, car le type délégué de l'événement ne correspond pas au type délégué d'un autre événement. Cette erreur se produit lorsque vous définissez plusieurs événements dans une interface et que vous tentez ensuite de les implémenter ensemble avec le même événement. Un événement peut implémenter deux ou plusieurs événements seulement si tous les événements implémentés sont déclarés à l'aide de la syntaxe **As** et spécifient le même type délégué.

ID d'erreur : BC31407

Pour corriger cette erreur

- Implémentez les événements séparément.

Voir aussi

Autres ressources

Événements en Visual Basic

L'événement '<nomévénement1>' ne peut pas implémenter l'événement '<nomévénement2>' pour l'interface '<interface>', car leurs types délégués '<délégué1>' et '<délégué2> ne correspondent pas

Event '<eventname1>' cannot implement event '<eventname2>' on interface '<interface>' because their delegate types '<delegate1>' and '<delegate2>' do not match

Visual Basic ne peut pas implémenter d'événement, car le type délégué de l'événement ne correspond pas au type délégué de l'événement dans l'interface. Cette erreur se produit lorsque vous définissez plusieurs événements dans une interface et que vous tentez ensuite de les implémenter ensemble avec le même événement. Un événement peut implémenter deux ou plusieurs événements seulement si tous les événements implémentés sont déclarés à l'aide de la syntaxe **As** et spécifient le même type délégué.

ID d'erreur : BC31423

Pour corriger cette erreur

- Implémentez les événements séparément.

  - ou -

- Définissez les événements dans l'interface à l'aide de la syntaxe **As** et spécifiez le même type délégué.

Voir aussi

Référence

Event, instruction

Delegate, instruction

Autres ressources

Événements en Visual Basic

La longueur du nom de l'évènement ne peut pas dépasser 1011 caractères

Event name length cannot exceed 1011 characters

Le nom ou identificateur de chaque événement déclaré avec une instruction **Event** est limité à 1011 caractères.

ID d'erreur : BC32204

Pour corriger cette erreur

- Réduisez la longueur de l'identificateur.

Voir aussi

Référence

Event, instruction

Concepts

Noms d'éléments déclarés

Les événements ne peuvent pas être déclarés avec un type délégué ayant un type de retour

Events cannot be declared with a delegate type that has a return type

Un délégué a été spécifié pour une procédure de fonction.

ID d'erreur : BC31084

Pour corriger cette erreur

- Spécifiez un délégué pour une procédure **Sub**.

Voir aussi

Autres ressources

Événements en Visual Basic

Les événements ne peuvent pas avoir de type de retour

Events cannot have a return type

Les événements peuvent accepter des arguments, mais ne peuvent pas retourner directement des valeurs.

ID d'erreur : BC30032

Pour corriger cette erreur

- Supprimez le type de retour de l'instruction **Event**.

Voir aussi

Référence

Event, instruction

Les événements déclarés avec une clause 'As' doivent avoir un type délégué

Events declared with an 'As' clause must have a delegate type

Un type autre que délégué a été spécifié.

ID d'erreur : BC31044

Pour corriger cette erreur

1. Supprimez la clause **As**.
2. Spécifiez un délégué avec la clause **As**.

Voir aussi

Référence

Event, instruction

Delegate, instruction

Autres ressources

Événements en Visual Basic

Délégués en Visual Basic

Les événements d'un module ne peuvent pas être déclarés '<spécificateur>'

Events in a Module cannot be declared '<specifier>'

Vous avez utilisé un spécificateur non valide pour un événement dans une instruction **Module**. Les modules ne peuvent jamais être instanciés, ne prennent pas en charge l'héritage et ne peuvent pas implémenter des interfaces.

ID d'erreur : BC30434

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Module, instruction

Les événements des interfaces ne peuvent pas être déclarés '<implements>'

Events in interfaces cannot be declared '<implements>'

Les événements déclarés dans des interfaces ne peuvent pas implémenter les événements d'autres interfaces.

ID d'erreur : BC30688

Pour corriger cette erreur

1. Supprimez l'instruction **Implements**.
2. Implémentez l'événement à l'intérieur d'une classe ou d'une structure.

Voir aussi

Référence

Interface, instruction (Visual Basic)

Les événements des variables WithEvents partagées ne peuvent pas être gérés par des méthodes non partagées

Events of shared WithEvents variables cannot be handled by non-shared methods

Une variable déclarée avec le modificateur **Shared** est une variable partagée. Une variable partagée identifie exactement un seul emplacement de stockage. Une variable déclarée avec le modificateur **WithEvents** garantit que le type auquel appartient la variable gère l'ensemble d'événements déclenchés par la variable. Quand une valeur est assignée à la variable, la propriété créée par la déclaration **WithEvents** déconnecte tout gestionnaire d'événements existant et raccorde le nouveau gestionnaire d'événements via la méthode **Add**.

ID d'erreur : BC30594

Pour corriger cette erreur

- Déclarez votre gestionnaire d'événements comme **Shared**.

Voir aussi

Référence

Shared (Visual Basic)

WithEvents

Concepts

Événements et gestionnaires d'événements

'Exit AddHandler', 'Exit RemoveHandler' et 'Exit RaiseEvent' ne sont pas valides

Message d'erreur
'Exit AddHandler', 'Exit RemoveHandler' et 'Exit RaiseEvent' ne sont pas valides. Utilisez 'Return' pour quitter les membres d'événement.
'Exit AddHandler', 'Exit RemoveHandler' and 'Exit RaiseEvent' are not valid. Use 'Return' to exit from event members.

L'instruction **Exit** ne peut pas être utilisée pour sortir des méthodes **AddHandler**, **RemoveHandler** ou **RaiseEvent** dans une déclaration **Custom Event**. Utilisez plutôt l'instruction **Return**, sans spécifier d'expression de retour, pour sortir de la méthode.

ID d'erreur : BC31111

Pour corriger cette erreur

- Remplacez l'instruction **Exit** par une instruction **Return**.

  Assurez-vous que l'instruction **Return** ne spécifie pas d'expression de retour.

Voir aussi

Référence
Event, instruction
AddHandler
RemoveHandler
RaiseEvent
Return, instruction (Visual Basic)
Autres ressources
Événements en Visual Basic

'Exit Do' ne peut apparaître qu'à l'intérieur d'une instruction 'Do'

'Exit Do' can only appear inside a 'Do' statement

Une instruction **Exit Do** apparaît à l'extérieur d'une boucle **Do**. **Exit Do** est valide uniquement entre une instruction **Do** et une instruction **Loop** correspondante.

ID d'erreur : BC30089

Pour corriger cette erreur

1. Vérifiez qu'une instruction **Do** valide précède **Exit Do** et qu'une instruction **Loop** valide apparaît après celle-ci.
2. Vérifiez que les autres structures de contrôle à l'intérieur de la boucle **Do** sont correctement terminées.

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)

'Exit For' ne peut apparaître qu'à l'intérieur d'une instruction 'For'

'Exit For' can only appear inside a 'For' statement

Une instruction **Exit For** apparaît à l'extérieur d'une boucle **For**. **Exit For** est valide uniquement entre une instruction **For** ou **For Each** et une instruction **Next** correspondante.

ID d'erreur : BC30096

Pour corriger cette erreur

1. Vérifiez qu'une instruction **For** ou **For Each** valide précède **Exit For** et qu'une instruction **Next** valide apparaît après celle-ci.
2. Vérifiez que les autres structures de contrôle à l'intérieur de la boucle **For** sont correctement terminées.

Voir aussi

Référence

For...Next, instruction (Visual Basic)
For Each...Next, instruction (Visual Basic)

'Exit Function' n'est pas valide dans 'Sub' ni 'Property'

'Exit Function' is not valid in a Sub or Property

Exit Function apparaît dans une procédure **Sub** ou **Property**. Une instruction **Exit** doit correspondre au bloc dans lequel elle apparaît.

ID d'erreur : BC30067

Pour corriger cette erreur

- Remplacez **Exit Function** par l'instruction **Exit Sub** ou **Exit Property**, selon le cas.

Voir aussi

Concepts

Sub, procédures
Function, procédures
Procédures Property

'Exit' doit être suivi de 'Sub', 'Function', 'Property', 'Do', 'For', 'While', 'Select' ou 'Try'

'Exit' must be followed by 'Sub', 'Function', 'Property', 'Do', 'For', 'While', 'Select', or 'Try'

Une instruction **Exit** contient un mot clé non valide. **Exit** doit spécifier le bloc à partir duquel le contrôle doit passer à l'instruction qui suit le bloc ; par exemple `End While`.

ID d'erreur : BC30240

Pour corriger cette erreur

- Ajoutez un mot clé valide à la suite de **Exit** ou supprimez l'instruction **Exit**.

Voir aussi

Référence

Exit, instruction (Visual Basic)

'Exit Operator' n'est pas valide. Utilisez 'Return' pour quitter un opérateur

'Exit Operator' is not valid. Use 'Return' to exit an Operator

Une instruction **Exit Operator** apparaît dans une procédure **Operator**.

Vous devez utiliser Return, instruction (Visual Basic) pour retourner à partir d'une procédure **Operator**.
Exit, instruction (Visual Basic) n'accepte pas le mot clé **Operator**, et l'instruction **End Operator** ne retourne pas le contrôle au code appelant.

ID d'erreur : BC33008

Pour corriger cette erreur

- Remplacez l'instruction **Exit Operator** par une instruction **Return**.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

'Exit Property' n'est pas valide dans 'Function' ni 'Sub'

'Exit Property' is not valid in a Function or Sub

Exit Property apparaît dans une procédure **Function** ou **Sub**. Une instruction **Exit** doit correspondre au bloc dans lequel elle apparaît.

ID d'erreur : BC30066

Pour corriger cette erreur

- Remplacez **Exit Property** par l'instruction **Exit Function** ou **Exit Sub**, selon le cas.

Voir aussi

Concepts

Sub, procédures
Function, procédures
Procédures Property

'Exit Select' ne peut apparaître qu'à l'intérieur d'une instruction 'Select'

'Exit Select' can only appear inside a 'Select' statement

Une instruction **Exit Select** apparaît à l'extérieur d'un bloc **Select**. **Exit Select** est valide uniquement entre une instruction **Select** ou **Select Case** et une instruction **End Select** correspondante.

ID d'erreur : BC30099

Pour corriger cette erreur

1. Vérifiez qu'une instruction **Select** ou **Select Case** valide précède **Exit Select** et qu'une instruction **End Select** valide apparaît après celle-ci.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **Select** sont correctement terminées.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

Les instructions 'Exit' ne sont pas valides dans la fenêtre Exécution

'Exit' statements are not valid in the Immediate window

Les instructions **Exit** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30713

Pour corriger cette erreur

- Supprimez les instructions **Exit** du code de débogage.

Voir aussi

Référence

Exit, instruction (Visual Basic)

End, instruction

Autres ressources

Débogage dans Visual Studio

'Exit Sub' n'est pas valide dans 'Function' ni 'Property'

'Exit Sub' is not valid in a Function or Property

Exit Sub apparaît dans une procédure **Function** ou **Property**. Une instruction **Exit** doit correspondre au bloc dans lequel elle apparaît.

ID d'erreur : BC30065

Pour corriger cette erreur

- Remplacez **Exit Sub** par l'instruction **Exit Function** ou **Exit Property**, selon le cas.

Voir aussi

Concepts

Sub, procédures
Function, procédures
Procédures Property

'Exit Try' ne peut apparaître qu'à l'intérieur d'une instruction 'Try'

'Exit Try' can only appear inside a 'Try' statement

Exit Try ne peut apparaître qu'à l'intérieur d'une instruction de bloc **Try**. Vous disposez d'une instruction **Exit Try** redondante ou votre instruction **Exit Try** apparaît en dehors des limites de son bloc **Try** correspondant.

ID d'erreur : BC30393

Pour corriger cette erreur

1. Recherchez et supprimez l'instruction **Exit Try** inutile.
2. Déplacez l'instruction **Exit Try** vers l'emplacement approprié dans votre code.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'Exit While' ne peut apparaître qu'à l'intérieur d'une instruction 'While'

'Exit While' can only appear inside a 'While' statement

Une instruction **Exit While** apparaît à l'extérieur d'un bloc **While**. **Exit While** est valide uniquement entre une instruction **While** et une instruction **End While** correspondante.

ID d'erreur : BC30097

Pour corriger cette erreur

1. Vérifiez qu'une instruction **While** valide précède **Exit While** et qu'une instruction **End While** valide apparaît après celle-ci.
2. Vérifiez que les autres structures de contrôle à l'intérieur du bloc **While** sont correctement terminées.

Voir aussi

Référence

While...End While, instruction (Visual Basic)

'Dim', 'Const', 'Public', 'Private', 'Protected', 'Friend', 'Shadows', 'ReadOnly' ou 'Shared' attendu

Expected one of 'Dim', 'Const', 'Public', 'Private', 'Protected', 'Friend', 'Shadows', 'ReadOnly' or 'Shared'

Un mot clé de déclaration manque dans l'instruction de déclaration. L'une des causes possibles est l'appel d'une méthode par une déclaration d'attribut.

ID d'erreur : BC30195

Pour corriger cette erreur

1. Vérifiez si une méthode est déclarée à l'intérieur d'une déclaration d'attribut.
2. Commencez l'instruction par le mot clé de déclaration approprié.

Voir aussi

Référence

Les tableaux ne peuvent pas être déclarés avec 'New'

Autres ressources

Éléments déclarés en Visual Basic

Une initialisation explicite n'est pas autorisée pour les tableaux déclarés avec des limites explicites

Explicit initialization is not permitted for arrays declared with explicit bounds

Les tableaux ne peuvent pas être initialisés s'ils sont déclarés être d'une taille spécifique.

ID d'erreur : BC30672

Pour corriger cette erreur

- Déclarez le tableau, puis initialisez-le séparément.
- Déclarez-le et initialisez-le en tant que tableau dynamique. Utilisez **ReDim** si nécessaire. Par exemple :

```
Dim A() As Integer = {0, 1, 2, 3}
ReDim Preserve A(3)
```

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Une initialisation explicite n'est pas autorisée avec plusieurs variables déclarées avec un spécificateur de type unique

Explicit initialization is not permitted with multiple variables declared with a single type specifier

L'initialisation n'est pas autorisée lorsque plusieurs variables sont déclarées sur la même ligne.

ID d'erreur : BC30671

Pour corriger cette erreur

1. Déclarez et initialisez chaque élément séparément.
2. Déclarez tous les éléments ensemble, puis initialisez chacun d'entre eux. Par exemple :

```
Dim x, b, i As Integer
x = 9 : b = 9 : i = 9 ' ":" is the same as a new line.
```

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic

Exposant non valide

Exponent is not valid

Vous avez tenté d'utiliser un exposant non valide avec un littéral en virgule flottante. Cette erreur se produit généralement si le type d'un littéral en virgule flottante ne correspond pas à la taille requise pour contenir le littéral en virgule flottante.

ID d'erreur : BC30495

Pour corriger cette erreur

- Vérifiez le type du littéral en virgule flottante et remplacez-le par un type valide.

Voir aussi

Tâches

Comment : calculer des valeurs numériques

L'expression appelle de manière récursive la propriété conteneur '<NomPropriété>'

Expression recursively calls the containing property '<propertyname>'

Une instruction dans la procédure **Set** d'une définition de propriété stocke une valeur dans le nom de la propriété.

L'approche recommandée pour stocker la valeur d'une propriété consiste à définir une variable **Private** dans le conteneur de la propriété et à l'utiliser dans les procédures **Get** et **Set**. La procédure **Set** doit ensuite stocker la valeur entrante dans cette variable **Private**.

La procédure **Get** se comportant comme une procédure **Function**, elle peut assigner une valeur au nom de la propriété et retourner le contrôle en utilisant l'instruction **End Get** Toutefois, l'approche recommandée consiste à inclure la variable **Private** comme valeur dans Return, instruction (Visual Basic).

La procédure **Set** se comporte comme une procédure **Sub**, qui ne retourne pas de valeur. Par conséquent, le nom de la procédure ou de la propriété n'a aucune signification particulière dans une procédure **Set**, et vous ne pouvez pas y stocker de valeur.

L'exemple suivant présente l'approche qui peut générer cette erreur, suivie de l'approche recommandée.

```
Public Class illustrateProperties
' The code in the following property causes this error.
    Public Property badProp() As Char
        Get
            Dim charValue As Char
            ' Insert code to update charValue.
            badProp = charValue
        End Get
        Set(ByVal Value As Char)
            ' The following statement causes this error.
            badProp = Value
            ' The value stored in the local variable badProp
            ' is not used by the Get procedure in this property.
        End Set
    End Property
' The following code uses the recommended approach.
    Private propValue As Char
    Public Property goodProp() As Char
        Get
            ' Insert code to update propValue.
            Return propValue
        End Get
        Set(ByVal Value As Char)
            propValue = Value
        End Set
    End Property
End Class
```

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42026

Pour corriger cette erreur

- Réécrivez la définition de la propriété pour utiliser l'approche recommandée, comme illustré dans l'exemple précédent.

Voir aussi

Référence

Property, instruction

Set, instruction (Visual Basic)

Concepts

Procédures Property

L'expression appelle de manière récursive l'opérateur conteneur '<SymboleOpérateur>'

Expression recursively calls containing Operator '<operatorsymbol>'

Une expression dans une procédure d'opérateur utilise l'opérateur défini. Cela se traduit par le fait que la procédure d'opérateur s'appelle elle-même en raison des types de données utilisés.

La procédure d'opérateur que vous définissez s'appelle elle-même si elle utilise le même opérateur avec l'un des éléments suivants :

- les mêmes opérandes pour lesquels vous définissez l'opérateur ;
- les opérandes des mêmes types de données pour lesquels vous définissez l'opérateur ; ou
- les opérandes des types de données qui s'étendent aux types de données pour lesquels vous définissez l'opérateur.

Un *appel récurrent* se produit lorsqu'une procédure s'appelle elle-même. Les appels récurrents peuvent entraîner une *boucle infinie*, dans laquelle le contrôle passe par le même ensemble d'instructions de manière répétée jusqu'à ce que votre application soit arrêtée en externe. Si votre code ne contient pas un ou plusieurs tests qui peuvent être utilisés pour arrêter la récursivité, une boucle infinie risque de se produire.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42004

Pour corriger cette erreur

- Si votre logique requiert que la procédure d'opérateur s'appelle elle-même, pensez à tester au moins une condition qui doit se produire à un moment donné, et utilisez ce test pour arrêter les appels récurrents.
- Si votre logique ne requiert pas que la procédure d'opérateur s'appelle elle-même, supprimez les appels récurrents ou remplacez-les par des instructions qui n'appellent pas leur propre procédure.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

L'expression ou l'instruction n'est pas valide dans les fenêtres de débogage

Expression or statement is not valid in debug windows

Vous ne pouvez pas utiliser les instructions **Try...Catch...Finally** dans un contexte de débogage.

ID d'erreur : BC30706

Pour corriger cette erreur

- Supprimez les instructions **Try...Catch...Finally** du code de débogage.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Débogage dans Visual Studio

L'expression de type '<NomType>' ne peut pas être convertie en 'Object' ou 'ValueType'

Expression of type '<typename>' cannot be converted to 'Object' or 'ValueType'

Une expression a pour valeur un type qui ne peut pas être converti (conversion boxing) par le Common Language Runtime (CLR).

La conversion *boxing* fait référence au traitement nécessaire pour convertir un type en **Object** ou, parfois, en ValueType. Le Common Language Runtime ne peut pas convertir certains types, par exemple ArgIterator et TypedReference.

Si vous n'avez pas utilisé **CType** ou **CObj** dans l'instruction contenant cette expression, Visual Basic a tenté une conversion implicite qui provoque cette erreur.

ID d'erreur : BC31394

Pour corriger cette erreur

1. Recherchez l'expression qui a pour valeur le type cité.
2. Recherchez la partie de votre instruction qui essaie de convertir le type cité.
3. Réécrivez l'instruction pour éviter la conversion boxing.

Voir aussi

Concepts

Conversions implicites et explicites

L'expression de type '<NomType1>' ne peut pas être de type '<NomType2>'

Expression of type '<typename1>' can never be of type '<typename2>'

Une expression **TypeOf**...**Is** teste une variable de référence d'objet à un type de données qu'elle ne peut pas contenir.

Dans certains cas, le compilateur peut déterminer qu'un test **TypeOf**...**Is** ne peut qu'échouer, par exemple s'il n'y a aucune relation d'héritage entre deux classes.

Le code suivant peut générer cette erreur.

```
Dim refVar as System.Windows.Forms.Form

If TypeOf refVar Is System.Array

End If
```

Form et Array étant des types sans aucune relation entre eux, le compilateur peut déterminer que l'expression **TypeOf**...**Is** retourne **False** pour toutes les valeurs de `refVar`.

ID d'erreur : BC31430

Pour corriger cette erreur

- Testez la variable pour un type de données réaliste ou supprimez complètement le test **TypeOf**...**Is**.

Voir aussi

Tâches

Comment : déterminer le type désigné par une variable objet

Référence

TypeOf, opérateur (Visual Basic)

L'expression est du type '<nomtype>', qui n'est pas un type de collection

Expression is of type '<typename>', which is not a collection type

La variable de groupe spécifiée dans une instruction **For Each** n'est pas un objet collection ou un tableau, et son type n'implémente pas l'interface IEnumerable. Le type doit prendre en charge le modèle de design de collection Visual Basic ou implémenter **IEnumerable**.

ID d'erreur : BC32023

Pour corriger cette erreur

- Déclarez la variable de groupe comme étant d'un type de classe qui prend en charge le design de collection Visual Basic ou implémente **IEnumerable**.

Voir aussi

Référence

For Each...Next, instruction (Visual Basic)

IEnumerable

Concepts

Classe de la collection Visual Basic

L'expression n'est pas un tableau ni une méthode et ne peut pas avoir de liste d'arguments

Expression is not an array or a method, and cannot have an argument list

Vous avez fourni une liste d'arguments pour une expression qui n'est ni un tableau, ni une méthode. Seuls les tableaux ou les méthodes peuvent avoir des listes d'arguments.

ID d'erreur : BC30471

Pour corriger cette erreur

- Supprimez la liste d'arguments.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

L'expression n'est pas une méthode

Expression is not a method

La cible de votre appel de méthode n'est pas une méthode.

ID d'erreur : BC30454

Pour corriger cette erreur

- Vérifiez que vous avez orthographié correctement le nom de la méthode.

Voir aussi

Concepts

Propriétés et méthodes
Méthodes de classe

Cette expression est une valeur et ne peut donc pas être la cible d'une assignation

Expression is a value and therefore cannot be the target of an assignment

Une instruction tente d'assigner une valeur à une expression. Vous ne pouvez assigner une valeur à une variable, une propriété ou un élément de tableau accessible en écriture qu'au moment de l'exécution . L'exemple suivant montre comment une telle erreur peut se produire.

```
Dim yesterday As Integer
ReadOnly maximum As Integer = 45
yesterday + 1 = DatePart(DateInterval.Day, Now)
' The preceding line is an ERROR because of an expression on the left.
maximum = 50
' The preceding line is an ERROR because maximum is declared ReadOnly.
```

Des exemples semblables pourraient s'appliquer à des propriétés et à des éléments de tableau.

Accès indirect. Un accès indirect via un type valeur peut également générer cette erreur. Prenez l'exemple de code suivant qui tente de définir la valeur de Point en y accédant indirectement par le biais de Location.

```
' Assume this code runs inside Form1.
Dim exitButton As New System.Windows.Forms.Button()
exitButton.Text = "Exit this form"
exitButton.Location.X = 140
' The preceding line is an ERROR because of no storage for Location.
```

La dernière instruction de l'exemple précédent échoue, car elle ne fournit qu'une allocation temporaire pour la structure **Point** retournée par la propriété **Location**. Une structure est un type valeur, et la structure temporaire n'est pas conservée après l'exécution de l'instruction. Le problème est résolu en déclarant et en utilisant une variable pour **Location**, ce qui a pour effet de créer une allocation plus permanente pour la structure **Point**. L'exemple de code suivant remplace la dernière instruction de l'exemple précédent.

```
Dim exitLocation as New System.Drawing.Point(140, exitButton.Location.Y)
exitButton.Location = exitLocation
```

ID d'erreur : BC30068

Pour corriger cette erreur

- Si l'instruction assigne une valeur à une expression, remplacez l'expression par une variable, une propriété ou un élément de tableau unique accessible en écriture.
- Si l'instruction effectue un accès indirect par le biais d'un type valeur (généralement une structure), créez une variable pour contenir le type valeur.
- Assignez la structure appropriée (ou autre type valeur) à la variable.
- Utilisez la variable pour accéder à la propriété pour lui assigner une valeur.

Voir aussi

Tâches

Procédures de dépannage

Concepts

Opérateurs et expressions en Visual Basic

Instructions d'assignation

L'expression a le type '<NomType>', qui est un type restreint et qui ne peut pas être utilisé pour accéder aux membres hérités de 'Object' ou 'ValueType'

Expression has the type '<typename>' which is a restricted type and cannot be used to access members inherited from 'Object' or 'ValueType'

Une expression a la valeur d'un type qui ne peut pas être boxed par le common language runtime (CLR) mais accède à un membre qui nécessite une conversion boxing.

Conversion boxing fait référence au traitement nécessaire pour convertir un type en **Object** ou, quelquefois, en ValueType. Le common language runtime ne peut pas convertir certains types de structure, par exemple ArgIterator, RuntimeArgumentHandle et TypedReference.

Cette expression tente d'utiliser le type restreint pour appeler une méthode héritée de Object ou de **ValueType**, comme GetHashCode ou ToString. Pour accéder à cette méthode, Visual Basic a tenté une conversion boxing implicite qui provoque cette erreur.

ID d'erreur : BC31393

Pour corriger cette erreur

1. Localisez l'expression qui a la valeur du type cité.
2. Localisez la partie de votre instruction qui essaie d'appeler la méthode héritée de **Object** ou de **ValueType**.
3. Réécrivez l'instruction pour éviter l'appel de méthode.

Voir aussi

Concepts

Conversions implicites et explicites

Expression attendue

Expression expected

Un opérateur, tel que **+** ou **=** requiert une expression pour son opérande de droite, mais aucune expression reconnaissable n'apparaît.

ID d'erreur : BC30201

Pour corriger cette erreur

- Fournissez une expression valide pour l'opérande de droite de l'opérateur.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic
Instructions d'assignation

Cette expression ne produit pas de valeur

Expression does not produce a value

Vous avez tenté d'utiliser une expression qui ne produit aucune valeur dans un contexte produisant des valeurs, tel que l'appel d'une instruction **Sub** dans un contexte où une fonction **Function** est attendue.

ID d'erreur : BC30491

Pour corriger cette erreur

- Remplacez l'expression par une expression produisant une valeur.

Voir aussi

Concepts

Types d'erreurs

Impossible d'évaluer l'expression à ce stade

Expression cannot be evaluated at this time

L'expression spécifiée n'a pas pu être évaluée dans le contexte actuel.

ID d'erreur : BC30736

Pour corriger cette erreur

- Modifiez la syntaxe de l'expression.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

Le champ '<nomchamp>' est d'un type non pris en charge

Field '<fieldname>' is of an unsupported type

Un champ est déclaré en tant que type qui ne peut pas être représenté par Visual Studio.

ID d'erreur : BC30656

Pour corriger cette erreur

1. Définissez le champ en tant que type pris en charge.
2. Développez un nouveau type qui décrit les données du champ.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Le champ ou la propriété '<nommembre>' est introuvable

Field or property '<membername>' is not found

L'attribut fait référence à un champ ou à une propriété non défini(e).

ID d'erreur : BC30661

Pour corriger cette erreur

1. Vérifiez l'orthographe du champ ou de la propriété.
2. Vérifiez que votre projet peut accéder à la classe d'attributs correcte et ajoutez-le aux références du projet si nécessaire.

Voir aussi

Référence

Imports, instruction

Le fichier <nomfichier> est introuvable

File <filename> could not be found

Le fichier spécifié est introuvable. Cela peut être dû à un chemin d'accès incorrect.

ID d'erreur : BC2001

Pour corriger cette erreur

- Vérifiez que le nom et le chemin d'accès du fichier sont corrects.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Le nom de fichier est déjà déclaré avec un GUID et une valeur de checksum différents

File name already declared with a different GUID and checksum value

Une tentative a été effectuée pour déclarer un fichier, mais le fichier existe déjà avec un GUID et une valeur de checksum différents.

ID d'erreur : BC42034

Pour corriger cette erreur

- Vérifiez que le nom du fichier et le chemin d'accès sont corrects.
- Supprimez le fichier redondant s'il n'est pas nécessaire.

Voir aussi

Concepts

Types d'erreurs

'Finally' ne peut apparaître qu'une seule fois dans une instruction 'Try'

'Finally' can only appear once in a 'Try' statement

L'instruction **Finally** est utilisée pour terminer un bloc **Try...Catch...Finally** ; elle ne peut donc apparaître qu'une seule fois à la fin du bloc.

ID d'erreur : BC30381

Pour corriger cette erreur

- Recherchez et supprimez l'instruction **Finally** redondante.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'Finally' ne peut pas apparaître en dehors d'une instruction 'Try'

'Finally' cannot appear outside a 'Try' statement

L'instruction **Finally** est utilisée pour terminer un bloc **Try...Catch...Finally** ; elle ne peut donc apparaître qu'une seule fois à la fin du bloc. Vous disposez d'une instruction **Finally** inutile ou votre instruction **Finally** apparaît en dehors des limites de son bloc **Try** correspondant.

ID d'erreur : BC30382

Pour corriger cette erreur

1. Recherchez et supprimez l'instruction **Finally** inutile.
2. Déplacez l'instruction **Finally** vers l'emplacement approprié dans votre code.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

'Finally' doit se terminer par une 'End Try' correspondante

'Finally' must end with a matching 'End Try'

Une instruction **Finally** apparaît dans votre code sans une instruction **End Try** correspondante. Les instructions **Finally** doivent se terminer par une instruction **End Try**.

ID d'erreur : BC30442

Pour corriger cette erreur

1. Supprimez l'instruction **Finally**.
2. Ajoutez une instruction **End Try** pour terminer le bloc.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Les instructions 'Finally' ne sont pas valides dans la fenêtre Exécution

'Finally' statements are not valid in the Immediate window

Les instructions **Finally** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30716

Pour corriger cette erreur

- Supprimez les instructions **Finally** du code de débogage.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Débogage dans Visual Studio

La première instruction du corps d'une méthode ne peut pas se trouver sur la même ligne que la déclaration de la méthode

First statement of a method body cannot be on the same line as the method declaration

Une instruction **Function**, **Sub**, **Get**, **Set** ou **Property** doit être seule sur une ligne de code source.

ID d'erreur : BC30040

Pour corriger cette erreur

1. Supprimez toute étiquette de ligne précédant la déclaration de procédure.
2. Déplacez toute instruction précédant la déclaration de procédure vers une ligne de code source précédente.
3. Déplacez toute instruction suivant la déclaration de procédure vers une ligne de code source suivante.

Voir aussi

Tâches

Comment : étiqueter des instructions

Concepts

Procédures dans Visual Basic

La première instruction de ce 'Sub New' doit être un appel à 'MyBase.New' ou 'MyClass.New' (plusieurs constructeurs accessibles sans paramètres)

Message d'erreur
La première instruction de ce 'Sub New' doit être un appel à 'MyBase.New' ou 'MyClass.New', car la classe de base '<base>' de '<dérivé>' a plusieurs 'Sub New' accessibles qu'il est possible d'appeler sans argument.
First statement of this 'Sub New' must be a call to 'MyBase.New' or 'MyClass.New' because base class '<base>' of '<derived>' has more than one accessible 'Sub New' that can be called with no arguments.

Un constructeur de classe ne fournit pas d'appel à un constructeur de classe de base et Visual Basic ne peut pas fournir d'appel implicite, car il n'est pas capable de déterminer le constructeur de classe de base qui doit être appelé.

ID d'erreur : BC32038

Pour corriger cette erreur

- Ajoutez un appel à un constructeur de classe de base **MyBase.New()**, ou à un autre constructeur de cette classe utilisant **MyClass.New()** ou **Me.New()**, sur la première ligne de ce constructeur.

Voir aussi

Référence
Utilisation de constructeurs et de destructeurs
MyBase
Concepts
Durée de vie d'un objet : création et destruction des objets

La première instruction de ce 'Sub New' doit être un appel à 'MyBase.New' ou 'MyClass.New' (aucun constructeur accessible sans paramètres)

Message d'erreur
La première instruction de ce 'Sub New' doit être un appel à 'MyBase.New' ou 'MyClass.New', car la classe de base '<nombase>' de '<nomdérivé>' n'a pas de 'Sub New' accessible pouvant être appelé sans argument.
First statement of this 'Sub New' must be a call to 'MyBase.New' or 'MyClass.New' because base class '<basename>' of '<derivedname>' does not have an accessible 'Sub New' that can be called with no arguments.

Dans une classe dérivée, chaque constructeur doit appeler un constructeur de classe de base (**MyBase.New**). Si la classe de base a un constructeur sans paramètres accessible aux classes dérivées, **MyBase.New** peut être appelé automatiquement. Dans le cas contraire, un constructeur de classe de base doit être appelé avec des paramètres et cela ne peut pas se faire automatiquement. Dans ce cas, la première instruction de chaque constructeur de classe dérivée doit appeler un constructeur paramétré sur la classe de base ou appeler un autre constructeur de la classe dérivée qui effectue un appel de constructeur de classe de base.

ID d'erreur : BC30148

Pour corriger cette erreur

- Appelez **MyBase.New** en fournissant les paramètres requis ou appelez un constructeur homologue qui effectuera cet appel.

Voir aussi

Référence
Utilisation de constructeurs et de destructeurs

La première instruction de ce 'Sub New' doit être un appel explicite à 'MyBase.New' ou 'MyClass.New', car le '<NomConstructeur>' dans la classe de base '<NomClasseDeBase>' de '<NomClasseDérivée>' est marqué comme obsolète.

First statement of this 'Sub New' must be an explicit call to 'MyBase.New' or 'MyClass.New' because the '<constructorname>' in the base class '<baseclassname>' of '<derivedclassname>' is marked obsolete.

Un constructeur de classe n'appelle pas explicitement un constructeur de la classe de base, et le constructeur implicite de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme une erreur.

Lorsqu'un constructeur de la classe dérivée n'appelle pas un constructeur de la classe de base, Visual Basic essaie de générer un appel implicite à un constructeur de la classe de base sans paramètre. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un appel implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30919

Pour corriger cette erreur

- Incluez un appel à **MyBase.New()** ou **MyClass.New()** comme première instruction de **Sub New** dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La première instruction de ce 'Sub New' doit être un appel explicite à 'MyBase.New' ou 'MyClass.New', car le '<NomConstructeur>' dans la classe de base '<NomClasseDeBase>' de '<NomClasseDérivée>' est marqué comme obsolète : '<MessageErreur>'

First statement of this 'Sub New' must be an explicit call to 'MyBase.New' or 'MyClass.New' because the '<constructorname>' in the base class '<baseclassname>' of '<derivedclassname>' is marked obsolete: '<errormessage>'

Un constructeur de classe n'appelle pas explicitement un constructeur de classe de base, et le constructeur de classe de base implicite est marqué avec l'attribut ObsoleteAttribute et la directive de le traiter comme une erreur.

Lorsqu'un constructeur de classe dérivée n'appelle pas de constructeur de classe de base, Visual Basic essaie de générer un appel implicite à un constructeur de classe de base sans paramètre. Si aucun constructeur n'est accessible dans la classe de base qui peut être appelée sans arguments, Visual Basic ne peut pas générer un appel implicite. Dans ce cas, le constructeur requis est marqué avec l'attribut **ObsoleteAttribute** et, de ce fait, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

ID d'erreur : BC30920

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Incluez un appel à **MyBase.New()** ou **MyClass.New()** comme première instruction du **Sub New** dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La première instruction de ce 'Sub New' doit être un appel explicite à 'MyBase.New' ou 'MyClass.New', car le '<NomConstructeur>' dans la classe de base '<NomClasseDeBase>' de '<NomClasseDérivée>' est marqué comme obsolète.

First statement of this 'Sub New' should be an explicit call to 'MyBase.New' or 'MyClass.New' because the '<constructorname>' in the base class '<baseclassname>' of '<derivedclassname>' is marked obsolete.

Un constructeur de classe n'appelle pas explicitement un constructeur de la classe de base, et le constructeur implicite de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme un avertissement.

Lorsqu'un constructeur de la classe dérivée n'appelle pas un constructeur de la classe de base, Visual Basic essaie de générer un appel implicite à un constructeur de la classe de base sans paramètre. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un appel implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41003

Pour corriger cette erreur

- Incluez un appel à **MyBase.New()** ou **MyClass.New()** comme première instruction de **Sub New** dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

La première instruction de ce 'Sub New' doit être un appel explicite à 'MyBase.New' ou 'MyClass.New', car le '<NomConstructeur>' dans la classe de base '<NomClasseDeBase>' de '<NomClasseDérivée>' est marqué comme obsolète : '<MessageErreur>'

First statement of this 'Sub New' should be an explicit call to 'MyBase.New' or 'MyClass.New' because the '<constructorname>' in the base class '<baseclassname>' of '<derivedclassname>' is marked obsolete: '<errormessage>'

Un constructeur de classe n'appelle pas explicitement un constructeur de la classe de base, et le constructeur implicite de la classe de base est marqué avec l'attribut ObsoleteAttribute et la directive pour le traiter comme un avertissement.

Lorsqu'un constructeur de la classe dérivée n'appelle pas un constructeur de la classe de base, Visual Basic essaie de générer un appel implicite à un constructeur de la classe de base sans paramètre. Si aucun constructeur accessible dans la classe de base ne peut être appelé sans arguments, Visual Basic ne peut pas générer un appel implicite. Dans ce cas, le constructeur requis étant marqué avec l'attribut **ObsoleteAttribute**, Visual Basic ne peut pas l'appeler.

Vous pouvez marquer les éléments de programmation comme n'étant plus utilisés en leur appliquant **ObsoleteAttribute**. Dans ce cas, vous pouvez affecter à la propriété IsError de l'attribut la valeur **True** ou **False**. Si vous affectez la valeur **True**, le compilateur traite une tentative d'utilisation de l'élément comme une erreur. Si vous affectez la valeur **False** ou laissez la valeur par défaut **False**, le compilateur émet un avertissement en cas de tentative d'utilisation de l'élément.

Par défaut, ce message est un avertissement, car la propriété **IsError** de **ObsoleteAttribute** a la valeur **False**. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41004

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Incluez un appel à **MyBase.New()** ou **MyClass.New()** comme première instruction de **Sub New** dans la classe dérivée.

Voir aussi

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'For Each' pour le type '<NomType>' est ambigu, car le type implémente plusieurs instanciations de 'System.Collections.Generic.IEnumerable(Of T)'

'For Each' on type '<typename>' is ambiguous because the type implements multiple instantiations of 'System.Collections.Generic.IEnumerable(Of T)'

Une instruction **For Each** spécifie une variable itérateur qui a plusieurs méthodes GetEnumerator.

La variable itérateur doit être d'un type qui implémente l'interface System.Collections.IEnumerable ou System.Collections.Generic.IEnumerable dans l'un des espaces de noms **Collections** du .NET Framework. Une classe peut implémenter plusieurs interfaces génériques construites à l'aide d'un argument de type différent pour chaque construction. Si une classe qui procède ainsi est utilisée pour la variable itérateur, cette dernière a plusieurs méthodes **GetEnumerator**. Dans ce cas, Visual Basic ne peut pas choisir quelle méthode appeler.

ID d'erreur : BC32096

Pour corriger cette erreur

- Utilisez DirectCast ou TryCast pour effectuer un cast du type de variable itérateur vers l'interface qui définit la méthode **GetEnumerator** que vous souhaitez utiliser.

Voir aussi

Référence

For Each...Next, instruction (Visual Basic)

Concepts

Exemples d'implémentation d'interface dans Visual Basic

Variable de contrôle de boucle For '<nomvariable>' déjà utilisée par une boucle For englobante

For loop control variable '<variablename>' already in use by an enclosing For loop

Toutes les boucles **For** imbriquées doivent utiliser des variables de contrôle uniques.

ID d'erreur : BC30069

Pour corriger cette erreur

- Changez un des noms de variables de contrôle.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

La variable de contrôle de boucle 'For' ne peut pas être de type '<type>'

'For' loop control variable cannot be of type '<type>'

Vous avez tenté d'utiliser une variable de contrôle de boucle dont le type n'est pas valide. Au début d'une boucle **For**, le point de départ, le point final et la valeur de pas sont évalués dans leur ordre d'apparition dans le texte. Les trois expressions doivent pouvoir être converties implicitement dans le type de la variable. Si la variable de la boucle **For** est de type **Object**, au moment de l'exécution, au moins une des expressions doit être d'un type numérique, et les trois expressions doivent pouvoir être forcées au type numérique le plus large d'entre elles.

ID d'erreur : BC30337

Pour corriger cette erreur

- Vérifiez le type de la variable de contrôle de boucle et remplacez-le par un type valide.

Voir aussi

Référence

For (Visual Basic)
Do...Loop, instruction (Visual Basic)
For...Next, instruction (Visual Basic)

'For' doit se terminer par un 'Next' correspondant

'For' must end with a matching 'Next'

Une instruction **For** apparaît sans instruction **Next** correspondante. Une instruction **Next** doit être utilisée pour terminer la boucle **For**.

ID d'erreur : BC30084

Pour corriger cette erreur

- Si cette boucle **For** fait partie d'un ensemble de boucles imbriquées, vérifiez que chaque boucle est correctement terminée.
- Ajoutez une instruction **Next** à la fin de la boucle **For**.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

'Get' est déjà déclaré

'Get' is already declared

Une instruction **Get** apparaît plusieurs fois dans un bloc de code. L'instruction **Get** déclare une procédure de propriété **Get** utilisée pour récupérer une valeur d'une propriété.

ID d'erreur : BC30443

Pour corriger cette erreur

- Supprimez l'instruction **Get** redondante.

Voir aussi

Référence

Get, instruction

L'accesseur 'Get' de la propriété '<NomPropriété>' n'est pas accessible

'Get' accessor of property '<propertyname>' is not accessible

Une instruction essaie de récupérer la valeur d'une propriété alors qu'elle n'a pas accès à la procédure **Get** de la propriété.

Si l'instruction Get est marquée avec un niveau d'accès plus restrictif que son instruction Property, une tentative de lecture de la valeur de la propriété peut échouer dans les cas suivants :

- L'instruction **Get** est marquée Private (Visual Basic) et le code appelant est situé à l'extérieur de la classe ou la structure dans laquelle la propriété est définie.
- L'instruction **Get** est marquée Protected (Visual Basic) et le code appelant ne figure pas dans la classe ou la structure dans laquelle la propriété est définie, ni dans une classe dérivée.
- L'instruction **Get** est marquée Friend (Visual Basic) et le code appelant ne figure pas dans l'assembly dans lequel la propriété est définie.

ID d'erreur : BC31103

Pour corriger cette erreur

- Si vous avez le contrôle du code source qui définit la propriété, vous devez déclarer la procédure **Get** avec le même niveau d'accès que la propriété proprement dite.
- Si vous n'avez pas le contrôle du code source qui définit la propriété, ou si vous devez limiter davantage le niveau d'accès de la procédure **Get** que celui de la propriété proprement dite, essayez de déplacer l'instruction qui lit la valeur de la propriété vers une zone de code qui offre un meilleur accès à la propriété.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Le type générique '<NomTypeGénérique>' ne peut pas être importé plus d'une fois

Generic type '<genericTypeName>' cannot be imported more than once

Une instruction Imports spécifie un type générique qui a déjà été importé avec un paramétrage de type différent.

Vous pouvez déclarer de nombreux types construits à partir d'un type générique, parce que vous ne redéfinissez pas le type générique en déclarant un type construit. Toutefois, l'importation d'un type générique à plusieurs reprises équivaut à plusieurs définitions.

ID d'erreur : BC32086

Pour corriger cette erreur

1. Si le fichier source contenant cette instruction **Imports** contient également une autre instruction **Imports** spécifiant le même type générique, supprimez l'une d'elles.
2. Si vous devez importer le même type générique avec des paramétrages de type différents, utilisez plusieurs fichiers sources.

Voir aussi

Concepts

Types génériques en Visual Basic

Le type de contrainte de paramètre générique <NomType> n'est pas conforme CLS

Generic parameter constraint type <typename> is not CLS-compliant

Un type générique est marqué comme `<CLSCompliant(True)>`, mais une contrainte sur l'un de ses paramètres de type spécifie un type qui est marqué comme `<CLSCompliant(False)>`, qui n'est pas marqué ou qui n'est pas qualifié car s'agissant d'un type non conforme.

Pour qu'un type soit conforme à la Spécification CLS (Common Language Specification) (CLS), il doit utiliser uniquement les types conformes CLS. Cette règle s'applique également aux contraintes des paramètres de type d'un type générique.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

Lorsque vous appliquez l'attribut CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40040

Pour corriger cette erreur

- Si le type générique doit prendre un paramètre de type contraint par ce type particulier, supprimez **CLSCompliantAttribute**. Le type ne peut pas être conforme CLS.
- Si le type générique doit être conforme CLS, remplacez le type de cette contrainte par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.

Voir aussi

Concepts

Types génériques en Visual Basic
Écriture d'un code conforme CLS

Les paramètres génériques utilisés comme types de paramètres optionnels doivent être contraints par classe

Generic parameters used as optional parameter types must be class constrained

Une procédure est déclarée avec un paramètre optionnel qui utilise un paramètre de type qui n'est pas contraint à être un type référence.

Vous devez toujours fournir une valeur par défaut pour chaque paramètre optionnel. Si le paramètre est d'un type référence, la valeur facultative doit être **Nothing**, qui est une valeur valide pour tout type référence. Toutefois, si le paramètre est d'un type valeur, ce type doit être un type de données élémentaire prédéfini par Visual Basic. En effet, un type valeur composite, tel qu'une structure définie par l'utilisateur, n'a aucune valeur par défaut valide.

Lorsque vous utilisez un paramètre de type pour un paramètre optionnel, vous devez garantir qu'il soit d'un type référence pour éviter d'utiliser un type valeur sans valeur par défaut valide. Cela signifie que vous devez contraindre le paramètre de type avec le mot clé **Class** ou avec le nom d'une classe spécifique.

ID d'erreur : BC32124

Pour corriger cette erreur

- Appliquez une contrainte au paramètre de type pour accepter uniquement un type référence, ou ne l'utilisez pas pour le paramètre optionnel.

Voir aussi

Référence

Liste de types
Class (Visual Basic)
Nothing (Visual Basic)

Concepts

Types génériques en Visual Basic
Paramètres facultatifs
Types valeur et types référence

Autres ressources

Structures constituées à partir de vos propres types de données

Les méthodes génériques ne peuvent pas utiliser une clause 'Handles'

Generic methods cannot use 'Handles' clause

Une procédure **Sub** générique contient une clause Handles dans sa déclaration.

Une clause **Handles** spécifie une liste d'événements gérés par la procédure **Sub**. Pour être un gestionnaire d'événements, la procédure **Sub** doit avoir la même signature que chaque événement qu'elle doit gérer. Une procédure générique peut être créée plusieurs fois, avec des signatures que Visual Basic ne peut pas prévoir au moment de la compilation. Par conséquent, Visual Basic ne peut pas garantir une signature qui correspond à celles des événements de la clause **Handles**.

ID d'erreur : BC32080

Pour corriger cette erreur

- Si la procédure **Sub** doit être générique, supprimez la clause **Handles** de sa déclaration. Utilisez l'instruction AddHandler pour associer ce gestionnaire d'événements à un événement.
- Si la procédure **Sub** doit utiliser la clause **Handles** pour associer des événements, supprimez la clause Of de sa déclaration. Vous devez utiliser une procédure non générique avec **Handles**.

Voir aussi

Concepts

Types génériques en Visual Basic
Événements et gestionnaires d'événements

Les méthodes génériques ne peuvent pas être exposées à COM

Generic methods cannot be exposed to COM

Un composant du .NET Framework qui contient une ou plusieurs procédures génériques est exporté vers un composant COM.

Le composant COM (Component Object Model) ne prend pas en charge les types génériques, et il ne peut pas interagir avec eux.

ID d'erreur : BC30943

Pour corriger cette erreur

- Supprimez la ou les procédures génériques du composant du .NET Framework, ou ne l'utilisez pas pour COM Interop.

Voir aussi

Concepts

Types génériques en Visual Basic

Autres ressources

COM Interop

Fonction sans clause 'As' ; type de retour Object pris par défaut

Function without an 'As' clause; return type of Object assumed

Une procédure **Function** ne spécifie pas de clause **As**.

Une clause **As** identifie un type de données à associer à un élément de programmation. Dans un Function, instruction (Visual Basic), elle spécifie le type de données de la valeur que la procédure **Function** retourne au code appelant. Si vous n'incluez pas de clause **As** dans l'instruction **Function**, le type de données retournées prend la valeur **Object** par défaut.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42021

Pour corriger cette erreur

- Incluez une clause **As** dans l'instruction **Function** pour spécifier le type de données retournées.

Voir aussi

Concepts

Function, procédures

L'évaluation de la fonction est désactivée, car une évaluation de fonction précédente a expiré

Message d'erreur
L'évaluation de la fonction est désactivée, car une évaluation de fonction précédente a expiré. Pour réactiver l'évaluation de la fonction, effectuez à nouveau le pas à pas détaillé ou relancez le débogage.
Function evaluation is disabled because a previous function evaluation timed out. To re-enable function evaluation, step again or restart debugging.

Dans le débogueur Visual Studio, une expression spécifie un appel de procédure, mais une autre évaluation a expiré.

Les causes possibles de l'expiration d'un appel de procédure sont, entre autres, une boucle infinie ou une *boucle sans fin*. Pour plus d'informations, consultez For...Next, instruction (Visual Basic).

Un cas particulier d'une boucle infinie est la *récursivité*. Pour plus d'informations, consultez Procédures récursives.

ID d'erreur : BC30957

Pour corriger cette erreur

1. Si possible, déterminez la précédente évaluation de fonction et la cause de son expiration. Sinon, cette erreur risque de se reproduire.
2. Exécutez à nouveau le débogueur pas à pas, ou arrêtez et relancez le débogage.

Voir aussi

Tâches
Comment : démarrer l'exécution
Référence
Expressions dans Visual Basic
Concepts
Comment : arrêter le débogage ou l'exécution
Vue d'ensemble du code pas à pas
Autres ressources
Débogage dans Visual Studio

La fonction '<NomProcédure>' ne retourne pas une valeur pour tous les chemins de code

Message d'erreur
La fonction '<NomProcédure>' ne retourne pas une valeur pour tous les chemins de code. Une exception de référence nulle pourrait se produire au moment de l'exécution lorsque le résultat est utilisé.
Function '<procedurename>' doesn't return a value on all code paths. A null reference exception could occur at run time when the result is used.

Une procédure **Function** peut au moins avoir un chemin d'accès par le biais de son code qui ne retourne pas de valeur.

Vous pouvez retourner une valeur d'une procédure **Function** en utilisant l'une des méthodes suivantes :

- assigner la valeur au nom de la procédure **Function**, puis exécuter une instruction **Exit Function** ;
- assigner la valeur au nom de la procédure **Function**, puis exécuter l'instruction **End Function** ;
- inclure la valeur dans un Return, instruction (Visual Basic).

Si le contrôle passe à **Exit Function** ou à **End Function** et que vous n'avez assigné aucune valeur au nom de la procédure, la procédure retourne la valeur par défaut du type de données retournées. Pour plus d'informations, consultez « Behavior » dans Function, instruction (Visual Basic).

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42105

Pour corriger cette erreur

- Vérifiez la logique de flux de votre contrôle et assurez-vous que vous assignez une valeur avant chaque instruction qui provoque un retour.

  Il est plus facile de garantir que chaque retour de la procédure retourne une valeur si vous utilisez toujours l'instruction **Return**. Si vous le faites, la dernière instruction avant **End Function** doit être une instruction **Return**.

Voir aussi

Référence
Function, instruction (Visual Basic)
Concepts
Function, procédures

Les noms qualifiés complets pour les types dans les espaces de noms ne peuvent pas dépasser <nombre> caractères

Fully qualified names for types in namespaces cannot be longer than <number> characters

Le nom qualifié complet du type ne peut pas être plus long que le nombre spécifié de caractères.

ID d'erreur : BC30031

Pour corriger cette erreur

- Utilisez un nom de type plus court.

Voir aussi

Concepts

Vérification du type dans Visual Basic

Une instruction 'Get' doit se terminer par un 'End Get' correspondant

'Get' statement must end with a matching 'End Get'

Les procédures de propriété **Get** doivent se terminer par une instruction **End Get**.

ID d'erreur : BC30631

Pour corriger cette erreur

- Vérifiez que la procédure de propriété **Get** se termine par une construction **End Get**.

Voir aussi

Référence

Property, instruction

Concepts

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

Les instructions 'Get' ne sont plus prises en charge

'Get' statements are no longer supported

Les instructions **Get** ne sont plus prises en charge. La fonctionnalité d'E/S sur fichier est disponible dans l'espace de noms **Microsoft.VisualBasic**.

Les instructions **Get** ne sont pas prises en charge pour les opérations de fichiers et ne peuvent être utilisées que dans la syntaxe de procédure de propriété.

ID d'erreur : BC30829

Pour corriger cette erreur

- Exécutez les opérations de fichiers à l'aide des membres de **System.IO**, **FileSystemObject** et des fonctions runtime Visual Basic.

Voir aussi

Référence

Get, instruction

Autres ressources

Traitement de lecteurs, de répertoires et de fichiers

Accès au fichier avec Visual Basic

Les instructions 'Get' ne sont plus prises en charge (Visual Basic)

'Get' statements are no longer supported (Visual Basic)

Les instructions **Get** ne sont plus prises en charge. La fonctionnalité d'E/S de fichier est normalement disponible dans l'espace de noms **Microsoft.VisualBasic**, mais la version ciblée du .NET Compact Framework ne la prend pas en charge.

ID d'erreur : BC30767

Pour corriger cette erreur

- Effectuez des opérations sur des fichiers à l'aide de fonctions définies dans l'espace de noms System.IO.

Voir aussi

Référence

Get, instruction

System.IO

Autres ressources

Accès au fichier avec Visual Basic

'Global' doit être suivi de '.' et d'un identificateur

'Global' must be followed by '.' and an identifier

Le mot clé Global apparaît dans un contexte autre que l'accès à un espace de noms.

Global a pour but d'autoriser votre code à accéder à un espace de noms racine à partir d'une structure d'espace de noms qui a bloqué cet espace de noms racine.

ID d'erreur : BC36000

Pour corriger cette erreur

- Si vous souhaitez accéder à un espace de noms racine, spécifiez-le après le mot clé **Global** et un point (**.**).

```
Dim keyInfo As Global.System.ConsoleKeyInfo
```

- Si vous ne souhaitez pas accéder à un espace de noms racine, supprimez le mot clé **Global**.

Voir aussi

Référence

Global

'Global' interdit dans les handles ; nom local attendu

'Global' not allowed in handles; local name expected

Une clause **Handles** doit faire référence à un événement local. Le mot clé **Global** fournit l'accès aux éléments de programmation globaux.

ID d'erreur : BC36002

Pour corriger cette erreur

- Modifiez la clause **Handles** pour faire référence à une instance locale de l'événement au lieu de l'instance globale.

Voir aussi

Référence

Global

Handles

Autres ressources

Événements en Visual Basic

'Global' n'est pas autorisé dans ce contexte ; identificateur attendu

'Global' not allowed in this context; identifier expected

Le mot clé **Global** est utilisé dans une instruction où il n'est pas autorisé.

Le mot clé **Global** vous permet d'accéder à un espace de noms défini à l'extérieur de la hiérarchie de l'espace de noms dans laquelle votre code doit être compilé. **Global** commence le chemin d'accès de qualification au niveau de l'espace de noms le plus éloigné de la bibliothèque de classes .NET Framework. Pour une illustration, consultez Global.

Certaines instructions, telles que **Imports** et **Namespace** sont indépendantes de l'espace de noms dans lequel votre code doit être compilé. Elles requièrent un chemin d'accès de qualification complet, commençant par l'espace de noms racine, tel que System ou Microsoft.VisualBasic. Dans ces instructions, le mot clé **Global** est superflu et n'est pas autorisé.

ID d'erreur : BC36001

Pour corriger cette erreur

- Supprimez le mot clé **Global** de l'instruction. Elle n'est pas nécessaire.

Voir aussi

Référence

Global
Imports, instruction
Namespace, instruction

Concepts

Références et l'instruction Imports

Les instructions 'GoSub' ne sont plus prises en charge

'GoSub' statements are no longer supported

L'instruction **GoSub** ne peut pas être utilisée pour appeler une procédure.

ID d'erreur : BC30814

Pour corriger cette erreur

- Appelez les procédures directement sans utiliser **GoSub**. Par exemple :

```
CalculateInterest(Amount, Rate, Time)
```

Voir aussi

Concepts

Procédures dans Visual Basic

'GoTo <nométiquette>' n'est pas valide, car '<nométiquette>' se trouve à l'intérieur d'une instruction 'For' ou 'For Each' qui ne contient pas cette instruction

'GoTo <labelname>' is not valid because '<labelname>' is inside a 'For' or 'For Each' statement that does not contain this statement

Les instructions **GoTo** sont limitées à des sauts à l'intérieur du bloc de code en cours.

ID d'erreur : BC30757

Pour corriger cette erreur

- Restructurez votre code de manière à ce que l'instruction **GoTo** et l'étiquette se trouvent à l'intérieur du bloc **For**.

Voir aussi

Référence

GoTo, instruction
For (Visual Basic)

'GoTo <nométiquette>' n'est pas valide, car '<nométiquette>' se trouve à l'intérieur d'une instruction 'SyncLock' qui ne contient pas cette instruction

'GoTo <labelname>' is not valid because '<labelname>' is inside a 'SyncLock' statement that does not contain this statement

Vous ne pouvez pas créer de branchement dans un bloc **SyncLock**.

ID d'erreur : BC30755

Pour corriger cette erreur

- Restructurez votre code de manière à ce que l'étiquette précède le bloc **SyncLock**.

Voir aussi

Référence

SyncLock, instruction

'GoTo <nométiquette>' n'est pas valide, car '<nométiquette>' se trouve à l'intérieur d'une instruction 'Try', 'Catch' ou 'Finally' qui ne contient pas cette instruction

'GoTo <labelname>' is not valid because '<labelname>' is inside a 'Try', 'Catch' or 'Finally' statement that does not contain this statement

Vous ne pouvez pas créer de branchement dans un bloc **Try...Catch...Finally**.

ID d'erreur : BC30754

Pour corriger cette erreur

- Restructurez votre code de manière à ce que l'étiquette précède le bloc **Try...Catch...Finally**.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

'GoTo <nométiquette>' n'est pas valide, car '<nométiquette>' se trouve à l'intérieur d'une instruction 'With' qui ne contient pas cette instruction

'GoTo <labelname>' is not valid because '<labelname>' is inside a 'With' statement that does not contain this statement

Les instructions **GoTo** sont limitées à des sauts à l'intérieur du bloc de code en cours.

ID d'erreur : BC30756

Pour corriger cette erreur

- Restructurez votre code de manière à ce que l'instruction **GoTo** et l'étiquette se trouvent à l'intérieur du bloc **With**.

Voir aussi

Référence

GoTo, instruction

'GoTo <ÉtiquetteDeLigne>' n'est pas valide, car '<ÉtiquetteDeLigne>' se trouve dans une instruction 'Using' qui ne contient pas cette instruction

'GoTo <linelabel>' is not valid because '<linelabel>' is inside a 'Using' statement that does not contain this statement

Une instruction **GoTo** à l'extérieur d'une construction **Using** essaie d'effectuer un branchement vers une étiquette de ligne à l'intérieur de la construction.

Il n'est pas possible d'effectuer un branchement depuis n'importe quel emplacement à l'extérieur de la construction **Using**...**End Using** vers n'importe quel emplacement à l'intérieur de cette dernière.

ID d'erreur : BC36009

Pour corriger cette erreur

- Remplacez l'étiquette de ligne de l'instruction **GoTo** par une étiquette qui n'est pas à l'intérieur d'une construction **For**...**Next**, **For Each**...**Next**, **SyncLock**...**End SyncLock**, **Try**...**Catch**...**Finally**, **With**...**End With** ou **Using**...**End Using**.

  - ou -

- Supprimez totalement l'instruction **GoTo**. La seule façon d'entrer une construction **Using**...**End Using** est de permettre que le contrôle passe à l'instruction **Using** elle-même.

Voir aussi

Référence

GoTo, instruction

Using, instruction (Visual Basic)

Les instructions 'GoTo' ne sont pas valides dans la fenêtre Exécution

'GoTo' statements are not valid in the Immediate window

L'instruction **GoTo** effectue un branchement et n'est pas autorisée dans un contexte de débogage.

ID d'erreur : BC30133

Pour corriger cette erreur

- N'émettez pas d'instruction **GoTo** dans la fenêtre **Exécution**.

Voir aussi

Référence

Fenêtre Exécution

La clause Handles requiert une variable WithEvents définie dans le type conteneur ou l'un de ses types de base

Handles clause requires a WithEvents variable defined in the containing type or one of its base types

Vous n'avez pas fourni de variable **WithEvents** dans votre clause **Handles**. Lorsque le mot clé **Handles** se trouve à la fin d'une déclaration de procédure, cette dernière gère les événements déclenchés par une variable objet déclarée à l'aide du mot clé **WithEvents**.

ID d'erreur : BC30506

Pour corriger cette erreur

- Fournissez la variable **WithEvents** nécessaire.

Voir aussi

Référence

Handles

Concepts

WithEvents et la clause Handles

Dans les classes, 'Handles' doit spécifier une variable 'WithEvents' ou 'MyBase', 'MyClass' ou 'Me' qualifiée par un identificateur unique

'Handles' in classes must specify a 'WithEvents' variable, 'MyBase', 'MyClass' or 'Me' qualified with a single identifier

Pour spécifier un gestionnaire d'événements, les instructions **Handles** doivent spécifier soit une variable objet qui a été déclarée à l'aide du mot clé **WithEvents**, soit un membre qualifié à l'aide du mot clé **MyBase**.

ID d'erreur : BC31412

Pour corriger cette erreur

1. Utilisez le modificateur **WithEvents** pour déclarer les variables à utiliser dans l'instruction **Handles**.
2. Spécifiez le nom d'un événement pour la classe en cours dans la classe de base.

Voir aussi

Référence

Handles

WithEvents

Autres ressources

Événements en Visual Basic

'Handles' dans les modules doit spécifier une variable 'WithEvents' qualifiée avec un unique identificateur

'Handles' in modules must specify a 'WithEvents' variable qualified with a single identifier

Pour spécifier un gestionnaire d'événements, les instructions **Handles** doivent spécifier une variable objet qui a été déclarée avec le mot clé **WithEvents**.

ID d'erreur : BC31418

Pour corriger cette erreur

- Utilisez le modificateur **WithEvents** pour déclarer des variables qui seront utilisées dans l'instruction **Handles**.

Voir aussi

Référence

Handles

WithEvents

Autres ressources

Événements en Visual Basic

'Handles' n'est pas valide pour les déclarations d'opérateur

'Handles' is not valid on operator declaration

Une instruction Operator spécifie le mot clé Handles.

Seule une procédure **Sub** peut gérer des événements. Une procédure **Operator** ne le peut pas. Pour plus d'informations sur les gestionnaires d'événements, consultez Comment : appeler un gestionnaire d'événements en Visual Basic.

Une procédure **Operator** requiert les mots clés **Public** et **Shared**, et un opérateur de conversion requiert le mot clé **Widening** ou **Narrowing**. Pour plus d'informations, consultez Procédures d'opérateur.

ID d'erreur : BC33003

Pour corriger cette erreur

- Si cette procédure doit gérer des événements, réécrivez-la en tant que procédure **Sub**.
- Si cette procédure doit définir un opérateur, supprimez le mot clé **Handles** de sa déclaration.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

Operator, instruction

Identificateur attendu

Identifier expected

Un élément de programmation qui n'est pas un nom d'élément déclaré reconnaissable apparaît lorsque le contexte requiert un nom d'élément. L'une des causes possibles est qu'un attribut a été spécifié à un endroit autre qu'au début de l'instruction.

ID d'erreur : BC30203

Pour corriger cette erreur

- Vérifiez que tous les attributs de l'instruction sont placés au début.
- Vérifiez que tous les noms d'éléments de l'instruction sont correctement orthographiés.

Voir aussi

Concepts

Noms d'éléments déclarés

Autres ressources

Attributs en Visual Basic

Identificateur trop long

Identifier is too long

Le nom ou l'identificateur de chaque élément de programmation est limité à 1023 caractères. De plus, un nom qualifié complet ne peut pas dépasser 1023 caractères. Cela signifie que la totalité de la chaîne d'identificateur (`<namespace>.<...>.<namespace>.<class>.<element>`) ne peut pas dépasser 1023 caractères, opérateurs d'accès au membre (**.**) inclus.

ID d'erreur : BC30033

Pour corriger cette erreur

- Réduisez la longueur de l'identificateur.

Voir aussi

Concepts

Noms d'éléments déclarés

'If' doit se terminer par un 'End If' correspondant

'If' must end with a matching 'End If'

Une instruction **If** apparaît sans instruction **End If** correspondante. Une instruction **End If** doit être utilisée pour terminer le bloc **If**.

ID d'erreur : BC30081

Pour corriger cette erreur

1. Si ce bloc **If** fait partie d'un ensemble de blocs **If** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Ajoutez une instruction **End If** à la fin du bloc **If**.

Voir aussi

Référence

If...Then...Else, instruction (Visual Basic)

'If', 'ElseIf', 'Else', 'EndIf' ou 'Const' attendu

'If', 'ElseIf', 'Else', 'End If', or 'Const' expected

Une ligne source commence par un caractère **#**, mais une directive de compilation conditionnelle valide ne suit pas immédiatement **#**. Les directives valides comprennent **#Const**, **#ExternalSource**, **#If**, **#Else**, **#ElseIf**, **#End If** et **#Region**.

ID d'erreur : BC30248

Pour corriger cette erreur

1. Assurez-vous que la directive de compilation conditionnelle est correctement orthographiée.
2. Vérifiez qu'il n'y a pas d'espace intermédiaire entre le caractère **#** et la directive.
3. Supprimez le caractère **#** ou ajoutez une directive valide immédiatement après celui-ci.

Voir aussi

Autres ressources

Directives (Visual Basic)

Option /noconfig ignorée, car elle était spécifiée dans un fichier réponse

Ignoring /noconfig option because it was specified in a response file

L'option **/noconfig** indique au compilateur de ne pas utiliser le fichier Vbc.rsp pour la compilation. Cependant, étant donné que le compilateur traite le fichier Vbc.rsp avant les autres fichiers réponse, il ne peut pas honorer l'option **/noconfig** dans un fichier réponse.

ID d'erreur : BC2025

Pour corriger cette erreur

1. Supprimez l'option **/noconfig** du fichier réponse.
2. Spécifiez l'option **/noconfig** lorsque vous appelez le compilateur de ligne de commande.

Voir aussi

Référence

/noconfig

@ (spécifier un fichier réponse) (Visual Basic)

Expression d'appel ou expression d'index non conforme

Illegal call expression or index expression

Une valeur entre parenthèses suit une expression qui n'est pas une procédure, une propriété ni un tableau.

Le code suivant peut générer ce message d'erreur.

```
Dim testVariable As Object = Nothing(1)
```

Étant donné que **Nothing** ne prend pas d'arguments ni d'index, vous ne pouvez pas l'utiliser avec des parenthèses.

ID d'erreur : BC32303

Pour corriger cette erreur

- Supprimez les parenthèses et les valeurs qu'elles contiennent.

Voir aussi

Tâches

Comment : appeler une procédure qui retourne une valeur
Comment : appeler une procédure qui ne retourne pas de valeur
Comment : placer une valeur dans un tableau
Comment : obtenir une valeur d'un tableau
Comment : placer une valeur dans une propriété
Comment : obtenir une valeur d'une propriété

Le type implémenté doit être une interface

Implemented type must be an interface

Une instruction **Implements** ne spécifie pas une interface. Une classe ne peut implémenter qu'une interface.

ID d'erreur : BC30232

Pour corriger cette erreur

1. Vérifiez que le nom d'interface est correctement orthographié.
2. Si l'instruction spécifie une classe, utilisez l'instruction **Inherits**.

Voir aussi

Référence

Implements, instruction

Inherits, instruction

Autres ressources

Héritage en Visual Basic

La classe d'implémentation '<NomClasseSous-jacente>' pour l'interface '<NomInterface>' n'est pas accessible dans ce contexte, car elle est '<NiveauAccès>'

Implementing class '<underlyingclassname>' for interface '<interfacename>' is not accessible in this context because it is '<accesslevel>'

Une interface est déclarée avec CoClassAttribute qui spécifie une classe sous-jacente, mais cette classe a un niveau d'accès qui empêche le code d'utilisation d'y accéder.

L'application de **CoClassAttribute** à une interface associe une classe sous-jacente à l'interface. Cela permet au code de créer directement un objet à partir de l'interface à l'aide d'une clause **New**.

Si le code qui utilise la clause **New** n'a pas accès à la classe sous-jacente, par exemple si la classe a la valeur **Private**, le compilateur génère cette erreur.

ID d'erreur : BC31109

Pour corriger cette erreur

1. Si vous disposez d'un contrôle de code source sur la classe sous-jacente, modifiez son niveau d'accès afin que le code d'utilisation puisse y accéder.
2. Si, pour une raison quelconque, vous ne pouvez pas modifier le niveau d'accès de la classe sous-jacente, supprimez la clause **New**. Vous ne pouvez pas créer directement un objet à partir de cette interface.

Voir aussi

Référence

New (Visual Basic)

CoClassAttribute

Concepts

Niveaux d'accès dans Visual Basic

La classe d'implémentation '<nomclasse>' pour l'interface '<nominterface>' est introuvable

Implementing class '<classname>' for interface <interfacename> cannot be found

Une classe d'implémentation de l'assembly d'interopérabilité n'est pas disponible.

ID d'erreur : BC31094

Pour corriger cette erreur

1. Vérifiez que la bibliothèque de types de l'objet COM est valide.
2. Utilisez l'importateur de bibliothèques de types (Tlbimp.exe) pour créer manuellement un assembly d'interopérabilité, puis ajoutez une référence à ce dernier à partir de votre projet.

Voir aussi

Référence

Implements, instruction

Type Library Importer (Tlbimp.exe)

Autres ressources

COM Interop

'Implements' n'est pas valide pour les déclarations d'opérateur

'Implements' is not valid on operator declaration

Une instruction Operator spécifie le mot clé Implements (Visual Basic).

Seule une procédure **Function** ou **Sub**, une propriété ou un événement peut implémenter un membre d'interface. Pour plus d'informations sur l'implémentation, consultez Exemples d'implémentation d'interface dans Visual Basic.

Une procédure **Operator** requiert les mots clés **Public** et **Shared**, et un opérateur de conversion requiert le mot clé **Widening** ou **Narrowing**. Pour plus d'informations, consultez Procédures d'opérateur.

ID d'erreur : BC33004

Pour corriger cette erreur

- Si cette procédure doit implémenter un membre d'interface, réécrivez-la en tant que procédure **Function** ou **Sub**, propriété ou événement.
- Si cette procédure doit définir un opérateur, supprimez le mot clé **Implements** de sa déclaration.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

Operator, instruction

'Implements' n'est pas valide dans les modules

'Implements' not valid in Modules

Une instruction ou une clause **Implements** apparaît à l'intérieur d'un module. Les modules ne peuvent pas implémenter des interfaces.

ID d'erreur : BC30231

Pour corriger cette erreur

- Supprimez l'instruction ou la clause **Implements** ou tapez à nouveau le module en tant que classe.

Voir aussi

Référence

Implements, instruction
Module, instruction

Les instructions 'Implements' doivent suivre toute instruction 'Inherits' et précéder toutes les déclarations dans une classe

'Implements' statement must follow any 'Inherits' statement and precede all declarations in a class

Une instruction **Implements** a été rencontrée à un emplacement non valide.

ID d'erreur : BC31053

Pour corriger cette erreur

- Placez les instructions **Implements** immédiatement après toute instruction **Inherits**, mais avant toute déclaration. Par exemple :

```
Class Derived
   Inherits Base
   Implements One
   Sub SubOne() Implements One.Method1
      ' Add code to implement the method.
   End Sub
End Class
```

Voir aussi

Référence

Implements (Visual Basic)

Autres ressources

Interfaces en Visual Basic

Conversion implicite de '<NomType1>' en '<NomType2>'

Implicit conversion from '<typename1>' to '<typename2>'

Une expression ou une instruction d'assignation accepte une valeur d'un type de données et la convertit en un autre type. Aucun mot clé de conversion n'étant utilisé, la conversion est *implicite*.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42016

Pour corriger cette erreur

- Si possible, utilisez des valeurs du même type de données pour que Visual Basic n'ait pas besoin d'effectuer de conversion.
- Utilisez **CType** ou un des autres mots clés de conversion afin que la conversion soit *explicite*.

Voir aussi

Référence

Fonctions de conversion de types de données

Concepts

Conversions implicites et explicites

Option Strict On interdit le passage du type '<NomType1>' au type '<NomType2>' lors de la recopie de la valeur du paramètre 'ByRef' '<NomParamètre>' dans l'argument correspondant.

Implicit conversion from '<typename1>' to '<typename2>' in copying the value of 'ByRef' parameter '<parametername>' back to the matching argument.

Une procédure est appelée avec un argument ByRef d'un type différent de celui de son paramètre correspondant.

Si vous passez un argument **ByRef**, Visual Basic copie parfois sa valeur dans une variable locale de la procédure au lieu de passer une référence. Dans ce cas, lors du retour de la procédure, Visual Basic doit copier la valeur de la variable locale dans l'argument du code appelant.

Si une valeur d'argument **ByRef** est copiée dans la procédure et que l'argument et le paramètre sont du même type, aucune conversion n'est nécessaire. Mais si les types sont différents, Visual Basic doit effectuer une conversion dans les deux sens. Étant donné que vous ne pouvez pas utiliser **CType** ou l'un des autres mots clés de conversion sur un argument ou un paramètre de procédure, une telle conversion est toujours implicite.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41999

Pour corriger cette erreur

- Si possible, utilisez un argument d'appel du même type que le paramètre de procédure. Ainsi, Visual Basic n'a pas besoin d'effectuer une conversion.
- Si vous devez appeler la procédure avec un type d'argument autre que le type du paramètre, sans retourner une valeur dans l'argument d'appel, affectez au paramètre la valeur ByVal au lieu de **ByRef**.

Voir aussi

Concepts

Procédures dans Visual Basic
Paramètres et arguments d'une procédure
Passage d'arguments par valeur et par référence
Conversions implicites et explicites

Les initialiseurs pour les membres de structure ne sont valides que pour les membres et constantes 'Shared'.

Initializers on structure members are valid only for 'Shared' members and constants

Une variable membre de structure a été initialisée en tant que partie de sa déclaration.

ID d'erreur : BC31049

Pour corriger cette erreur

- Utilisez une constante plutôt qu'une variable.
- Ajoutez un constructeur paramétré à la structure. Par exemple :

```
Structure TestStruct
   Public t As Integer
   Sub New(ByVal Tval As Integer)
      t = Tval
   End Sub
End Structure
```

Voir aussi

Tâches

Comment : déclarer une structure

Autres ressources

Constantes et énumérations en Visual Basic

Erreur d'initialisation

Initialization error

Une erreur s'est produite lors de l'initialisation du programme. Cette erreur peut résulter de l'annulation d'une installation en cours d'exécution.

ID d'erreur : BC2000

Pour corriger cette erreur

- Tentez à nouveau d'effectuer l'installation.

Voir aussi

Concepts

Types d'erreurs

Les instructions 'Inherits' doivent précéder toutes les déclarations dans une interface

'Inherits' statements must precede all declarations in an interface

Une instruction **Inherits** doit être la première ligne non vide et sans commentaire dans une définition de classe.

ID d'erreur : BC30357

Pour corriger cette erreur

- Déplacez votre instruction **Inherits** de sorte qu'elle précède toutes les déclarations.

Voir aussi

Référence

Inherits, instruction

L'instruction 'Inherits' doit précéder toutes les déclarations dans une classe

'Inherits' statement must precede all declarations in a class

Les instructions **Inherits** doivent figurer au début des définitions de classes.

ID d'erreur : BC30683

Pour corriger cette erreur

- Déplacez l'instruction **Inherits** au début de votre classe.

Voir aussi

Référence

Inherits, instruction

Autres ressources

Héritage en Visual Basic

'Inherits' n'est pas valide dans les modules

'Inherits' not valid in Modules

Une instruction **Inherits** apparaît à l'intérieur d'un module. Les modules ne peuvent pas hériter de classes.

ID d'erreur : BC30230

Pour corriger cette erreur

- Supprimez l'instruction **Inherits** ou tapez à nouveau le module en tant que classe.

Voir aussi

Référence

Inherits, instruction
Module, instruction

'Inherits' ne peut apparaître qu'une seule fois dans une instruction 'Class' et ne peut spécifier qu'une classe

'Inherits' can appear only once within a 'Class' statement and can only specify one class

Plusieurs instructions **Inherits** apparaissent dans la même classe ou une instruction **Inherits** spécifie plusieurs classes. Une classe peut hériter d'une seule classe de base.

ID d'erreur : BC30121

Pour corriger cette erreur

- Supprimez toutes les instructions **Inherits** supplémentaires et assurez-vous que l'instruction **Inherits** restante spécifie une seule classe de base.

Voir aussi

Référence

Inherits, instruction

Concepts

Éléments fondamentaux de l'héritage

Hériter de 'System.<nomclasse>' n'est pas valide

Inheriting from 'System.<classname>' is not valid

Une classe ne peut pas hériter de **System.Array**, **System.Delegate**, **System.Enum** ou **System.ValueType**.

ID d'erreur : BC30015

Pour corriger cette erreur

- Supprimez l'instruction **Inherits** ou modifiez-la afin qu'elle hérite d'une classe de base valide.

Voir aussi

Concepts

Éléments fondamentaux de l'héritage
Déclaration des variables objets

Les informations pour le type de '<nomtype>' n'ont pas été chargées dans le runtime

Information for the type of '<typename>' has not been loaded into the runtime

Un type n'ayant pas été chargé par le runtime a été référencé.

ID d'erreur : BC30750

Pour corriger cette erreur

1. Restructurez le code de manière à ce que le type soit défini dans la portée actuelle.
2. Vérifiez que le membre est défini et que vous avez correctement orthographié son nom.
3. Essayez d'accéder à l'un des membres déclarés dans le module. Dans certains cas, l'environnement de débogage ne peut pas localiser les membres car les modules dans lesquels ils sont déclarés ne sont pas chargés.

Voir aussi

Autres ressources

Débogage dans Visual Studio

La contrainte indirecte '<contrainte1>' obtenue de la contrainte de paramètre de type '<ParamètreType1>' est en conflit avec la contrainte indirecte' <contrainte2>' obtenue de la contrainte de paramètre de type '<ParamètreType2>'

Indirect constraint '<constraint1>' obtained from the type parameter constraint '<typeparameter1>' conflicts with the indirect constraint '<constraint2>' obtained from the type parameter constraint '<typeparameter2>'

Un type générique est déclaré avec des contraintes incompatibles en raison d'une combinaison de contraintes indirectes.

L'instruction suivante peut générer ce message d'erreur.

```
Public Class testClass(Of t1 As {t2, t3}, t2 As Structure, t3 As Class)
```

Les contraintes indirectes `Structure` et `Class` créent un conflit pour le paramètre de type `t1`, car la contrainte **Structure** requiert que l'argument de type correspondant soit un type valeur, tandis que **Class** requiert qu'il soit un type référence.

ID d'erreur : BC32109

Pour corriger cette erreur

- Modifiez les contraintes du paramètre de type pour éviter les contraintes incompatibles.

Voir aussi

Référence

Liste de types
Structure (Visual Basic)
Class (Visual Basic)

Concepts

Types génériques en Visual Basic
Types valeur et types référence

La contrainte indirecte '<contrainte1>' obtenue à partir de la contrainte de paramètre de type '<TypeParamètre1>' est en conflit avec la contrainte '<contrainte2>'

Indirect constraint '<constraint1>' obtained from the type parameter constraint '<typeparameter1>' conflicts with the constraint '<constraint2>'

Un type générique est déclaré avec des contraintes incompatibles en raison d'une combinaison de contraintes directes et indirectes.

L'instruction suivante peut générer ce message d'erreur.

```
Public Class testClass(Of t1 As {t2, Class}, t2 As Structure)
```

La contrainte indirecte `Structure` et la contrainte directe `Class` créent un conflit pour le paramètre de type `t1`, car la contrainte **Structure** requiert que l'argument de type correspondant soit un type valeur, tandis que la contrainte **Class** requiert qu'il soit un type référence.

ID d'erreur : BC32111

Pour corriger cette erreur

- Modifiez les contraintes du paramètre de type pour éviter les contraintes incompatibles.

Voir aussi

Référence

Liste de types
Structure (Visual Basic)
Class (Visual Basic)

Concepts

Types génériques en Visual Basic
Types valeur et types référence

Une référence indirecte est établie à l'assembly <NomAssembly> version <NuméroDernièreVersion>, qui contient '<NomType>'

Message d'erreur
Une référence indirecte est établie à l'assembly <NomAssembly> version <NuméroDernièreVersion>, qui contient '<NomType>'. Ce projet fait référence à une version antérieure de <NomAssembly> version <NuméroDernièreVersion>. Pour utiliser '<NomType>', vous devez remplacer la référence à <NomAssembly> par la version <NuméroDernièreVersion> ou ultérieure.
Indirect reference is being made to assembly <assemblyname> version <laterversionnumber>, which contains '<typename>'. This Project references a prior version of <assemblyname> version <earlierversionnumber>. To use '<typename>', you must replace the reference to <assemblyname> with version <laterversionnumber> or higher.

Une expression crée une référence indirecte à un autre projet, qui contient une référence à une version antérieure du même assembly.

En général, vous devez utiliser uniquement la version la plus récente d'un assembly.

ID d'erreur : BC32207

Pour corriger cette erreur

1. Utilisez le nom de type mentionné pour déterminer le projet qui fait également référence au même assembly.
2. Déterminez la version de l'assembly référencée par l'autre projet, et modifiez votre projet pour faire référence à la même version.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Dépannage de références rompues
Concepts
Références de projet

L'index '<numéroindex>' pour la dimension '<numérodimension>' est hors limites

Index '<indexnumber>' for dimension '<dimensionnumber>' is out of range

Un des index est en dehors de la plage définie.

ID d'erreur : BC30702

Pour corriger cette erreur

- Changez la logique de la procédure de manière à ce que les limites du tableau ne soient pas dépassées.
- Redéfinissez le tableau afin de permettre une plus grande plage de valeurs.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Utilisation inappropriée du mot clé <MotClé> dans un module

Les modules ne contiennent pas d'instances, ne prennent pas en charge l'héritage ou n'implémentent pas d'interfaces. C'est la raison pour laquelle les mots clés suivants ne s'appliquent pas à une déclaration de module :

- MustInherit
- NotInheritable
- Overloads
- Private (Visual Basic)
- Protected (Visual Basic)
- Shadows
- Shared (Visual Basic)
- Static (Visual Basic)

Les seuls mots clés pris en charge dans l'instruction Module sont Public (Visual Basic) et Friend (Visual Basic).

De plus, vous ne pouvez pas utiliser l'instruction Implements (Visual Basic) ouInherits dans le bloc d'instructions du module.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42028

Pour corriger cette erreur

- Si cet élément de programmation doit être un module, utilisez uniquement le mot clé **Public** ou **Friend** dans sa déclaration. Un module prend par défaut la valeur **Friend** si vous ne spécifiez pas son niveau d'accès.
- Si vous comptez créer des instances de cet élément de programmation, déclarez-le comme une classe. Vous pouvez ensuite utiliser les mots clés qui s'appliquent à une déclaration de classe.

Voir aussi

Référence

Class, instruction (Visual Basic)

Des instructions 'Imports' doivent précéder toutes les déclarations

'Imports' statements must precede any declarations

Une instruction **Imports** suit une instruction de déclaration dans un fichier source.

L'instruction **Imports** importe les noms d'espaces de noms des projets et des assemblys référencés, ainsi que les noms d'espaces de noms définis dans le même projet que celui dans lequel elle apparaît. Les instructions **Imports** doivent être placées dans un fichier source avant les références aux identificateurs.

ID d'erreur : BC30465

Pour corriger cette erreur

- Déplacez l'instruction **Imports** en haut du fichier source, avant les instructions de déclaration.

Voir aussi

Référence

Imports, instruction

L'alias '<nom>' pour Imports est en conflit avec '<nom2>' déclaré dans l'espace de noms racine

Imports alias '<name>' conflicts with '<name2>' declared in the root namespace

Le nom de l'alias utilisé dans l'instruction **Imports** est en conflit avec un autre nom déclaré dans l'espace de noms racine.

ID d'erreur : BC31403

Pour corriger cette erreur

- Choisissez un nom unique pour l'alias utilisé par l'instruction **Imports**.

Voir aussi

Référence

Imports, instruction

Échec de l'importation du type '<nomtype>' à partir de l'assembly ou du module '<nom>'

Import of type '<typename>' from assembly or module '<name>' failed

Vérifiez que toutes les bibliothèques référencées sont compatibles les unes avec les autres.

ID d'erreur : BC31091

Pour corriger cette erreur

- Vérifiez que les fichiers qui contiennent le type référencé ne sont pas endommagés.

Voir aussi

Référence

Imports, instruction

Concepts

Références et l'instruction Imports

La variable implicite '<NomVariable>' n'est pas valide en raison de '<message>'

Implicit variable '<variablename>' is invalid because of '<message>'

Une variable implicite non valide a été générée pendant le débogage.

ID d'erreur : BC36016

Pour corriger cette erreur

1. Examinez le message cité pour identifier la cause de l'erreur et trouver une solution éventuelle.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Concepts

Aide auprès des Services du Support Technique Microsoft (Visual Studio)*

Autres ressources

Support technique et accessibilité

Erreur interne du compilateur : le générateur de code a reçu une entrée incorrecte

Internal compiler error: code generator received malformed input

Le compilateur Visual Basic n'a pas réussi à interpréter un de ses fichiers écrits en interne.

ID d'erreur : BC30011

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
3. Si l'erreur persiste après la réinstallation, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Erreur interne du compilateur

Internal compiler error

Une erreur interne s'est produite lors de l'évaluation de l'expression.

ID d'erreur : BC30747

Pour corriger cette erreur

- Redémarrez Visual Studio.

Voir aussi

Autres ressources

Support technique

Les paramètres 'InterfaceId' et 'EventsId' pour 'Microsoft.VisualBasic.ComClassAttribute' sur '<nomtype>' ne peuvent pas avoir la même valeur

'InterfaceId' and 'EventsId' parameters for 'Microsoft.VisualBasic.ComClassAttribute' on '<typename>' cannot have the same value

Un bloc d'attributs **COMClassAttribute** spécifie le même GUID pour l'interface et l'événement de création. Si ces identificateurs sont tous les deux fournis, ils doivent être différents. Ils doivent également être différents de l'identificateur de classe.

Un GUID se compose de 16 octets, dont les huit premiers sont numériques et les huit derniers sont binaires. Il est généré par les utilitaires Microsoft, tels que uuidgen.exe et est unique.

ID d'erreur : BC32507

Pour corriger cette erreur

1. Déterminez les GUID corrects nécessaires pour identifier l'interface et l'événement de création pour l'objet COM.
2. Assurez-vous que les chaînes du GUID présentées au bloc d'attributs **COMClassAttribute** ont été copiées correctement.

Voir aussi

Référence

ComClassAttribute, classe

Concepts

Attributs utilisés dans Visual Basic

Application des attributs

'Interface' doit se terminer par un 'End Interface' correspondant

'Interface' must end with a matching 'End Interface'

Une instruction **Interface** apparaît sans instruction **End Interface** correspondante. Vous devez utiliser une instruction **End Interface** pour terminer la définition d'interface.

ID d'erreur : BC30253

Pour corriger cette erreur

1. Vérifiez que les membres **Interface** sont correctement mis en forme.
2. Ajoutez une instruction **End Interface** à la fin de la définition d'interface.

Voir aussi

Référence

Interface, instruction (Visual Basic)

Les membres d'interface doivent être des méthodes, des propriétés, des événements ou des définitions de types

Interface members must be methods, properties, events, or type definitions

Seuls les types imbriqués, les méthodes, les propriétés et les événements peuvent être des membres d'une interface.

ID d'erreur : BC30602

Pour corriger cette erreur

- Vérifiez que l'élément est une méthode, une propriété, un événement ou un type imbriqué, tel qu'une énumération.

Voir aussi

Concepts

Définition des interfaces

Une interface ne peut hériter que d'une autre interface

Interface can inherit only from another interface

Vous avez tenté d'hériter d'un élément autre qu'une interface.

ID d'erreur : BC30354

Pour corriger cette erreur

- Vérifiez que la ou les interfaces de base à partir desquelles vous tentez d'hériter sont réellement des interfaces.

Voir aussi

Référence

Interface, instruction (Visual Basic)

Concepts

Éléments fondamentaux de l'héritage

L'interface '<nominterface>' n'est pas implémentée par cette classe

Interface '<interfacename>' is not implemented by this class

Un membre de cette classe ou structure tente d'implémenter un membre d'une interface que la classe ou la structure n'implémente pas.

ID d'erreur : BC31035

Pour corriger cette erreur

- Ajoutez une instruction **Implements** au début de la classe ou de la structure afin qu'elle implémente l'interface appropriée.
- Supprimez le mot clé **Implements** du membre qui provoque cette erreur.

Voir aussi

Référence

Implements (Visual Basic)

Implements, instruction

Autres ressources

Interfaces en Visual Basic

L'interface '<NomInterface>' ne peut pas hériter d'elle-même : <message>

Interface '<interfacename>' cannot inherit from itself: <message>

Une instruction Inherits dans une définition d'interface spécifie sa propre interface.

Une interface peut hériter d'une autre interface, qui lui fournit tous les membres de l'interface dont elle hérite. Par conséquent, elle ne doit pas redéfinir ces membres. Cette interface est appelée *interface dérivée*, et l'interface dont elle hérite est appelée *interface de base*.

Cela n'a pas de sens qu'une interface hérite d'elle-même, parce qu'elle possède déjà tous ses propres membres.

ID d'erreur : BC30296

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom de l'interface dans l'instruction **Inherits**.
2. Si vous ne comptez pas hériter d'une autre interface, supprimez l'intégralité de l'instruction **Inherits**.
3. Examinez le message cité pour obtenir des suggestions.

Voir aussi

Autres ressources

Héritage en Visual Basic
Interfaces en Visual Basic

L'interface '<nominterface>' ne peut pas être indexée, car elle n'a pas de propriété par défaut

Interface '<interfacename>' cannot be indexed because it has no default property

Les expressions d'index doivent produire soit une valeur de type tableau ou dont le type possède un ensemble de propriétés par défaut surchargées, soit un ensemble de propriétés surchargées. Une interface ne peut avoir qu'une méthode ou propriété par défaut. Ceci signifie qu'une interface peut soit hériter d'une interface contenant une méthode ou une propriété par défaut, soit que son bloc de définition peut contenir une méthode ou propriété qui est marquée comme étant par défaut.

ID d'erreur : BC30547

Pour corriger cette erreur

- Spécifiez une propriété par défaut dans l'interface.

Voir aussi

Référence

Interface, instruction (Visual Basic)

L'interface '<nominterface>' ne peut être implémentée qu'une seule fois par ce type

Interface '<interfacename>' can be implemented only once by this type

L'interface spécifiée est déjà implémentée par ce type et ne peut pas l'être à nouveau.

ID d'erreur : BC31033

Pour corriger cette erreur

- Implémentez cette interface une seule fois en utilisant ce type.

Voir aussi

Référence

Implements (Visual Basic)

Autres ressources

Interfaces en Visual Basic

Constante Integer attendue

Integer constant expected

Une directive **#ExternalSource** apparaît avec un deuxième argument qui n'est pas un littéral entier. Seul un littéral entier est valide dans ce contexte. Une constante nommée ou un membre d'énumération n'est pas valide.

ID d'erreur : BC30204

Pour corriger cette erreur

1. Utilisez une constante nommée ou un membre d'énumération à la place d'un littéral.
2. Fournissez un littéral entier comme deuxième argument de la directive **#ExternalSource**.

Voir aussi

Référence

#ExternalSource, directive

'IsNot' requiert des opérandes qui ont des types référence, mais cet opérande a le type valeur '<NomType>'

'IsNot' requires operands that have reference types, but this operand has the value type '<typename>'.

Une expression utilise l'opérateur IsNot avec au moins une opérande de type valeur.

L'opérateur **IsNot** détermine si deux références d'objet font référence à des objets différents. Il compare les valeurs de pointeur des types référence et est sans signification par rapport aux types valeur.

ID d'erreur : BC31419

Pour corriger cette erreur

- Si vous avez l'intention de comparer les valeurs de deux types valeur, utilisez l'opérateur de comparaison **=** ou **<>**.
- Si vous avez l'intention de comparer les pointeurs de deux types référence, vérifiez que vous utilisez les références d'objet pour les deux opérandes. Vous pouvez utiliser des variables ou des éléments de référence, tels que le mot clé Me qui se comporte comme les variables de référence.

Voir aussi

Tâches

Comment : déterminer si deux objets sont identiques

Concepts

Opérateurs de comparaison en Visual Basic

L'opérande 'IsNot' du type '<NomParamètreDeType>' ne peut être comparé qu'à 'Nothing', car '<NomParamètreDeType>' est un paramètre de type sans contrainte de classe

'IsNot' operand of type '<typeparametername>' can be compared only to 'Nothing' because '<typeparametername>' is a type parameter with no class constraint

Un paramètre de type est utilisé comme opérande pour l'opérateur IsNot lorsqu'il est défini sans le mot clé Class (Visual Basic) ou sans un nom de classe spécifique dans sa liste de contraintes.

IsNot compare deux types référence pour déterminer s'ils pointent vers différentes instances d'objet en mémoire. Il ne peut pas accepter un opérande qui n'est pas un type référence à moins que l'autre opérande soit Nothing (Visual Basic).

ID d'erreur : BC32097

Pour corriger cette erreur

- Si vous souhaitez que l'argument de type fourni à ce paramètre de type soit toujours un type référence, ajoutez le mot clé **Class** ou un nom de classe spécifique à la liste des contraintes pour le paramètre de type.
- Si vous ne souhaitez pas que l'argument de type fourni à ce paramètre de type soit toujours un type référence, supprimez-le de l'expression **IsNot**. Vous ne pouvez pas le comparer à d'autres types référence à l'aide de l'opérateur **IsNot**.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic
Types valeur et types référence
Opérateurs de comparaison en Visual Basic

'Is' requiert des opérandes ayant des types référence, mais cet opérande a le type valeur '<nomtype>'

'Is' requires operands that have reference types, but this operand has the value type '<typename>'

L'opérateur de comparaison **Is** détermine si deux variables objets font référence à la même instance. Cette comparaison n'est pas définie pour les types valeur.

ID d'erreur : BC30020

Pour corriger cette erreur

- Utilisez l'opérateur de comparaison arithmétique approprié ou l'opérateur **Like** pour comparer deux types valeur.

Voir aussi

Référence

Is, opérateur (Visual Basic)
Like, opérateur
Opérateurs de comparaison (Visual Basic)

L'opérande 'Is' de type '<NomParamètreDeType>' ne peut être comparé qu'à 'Nothing', car '<NomParamètreDeType>' est un paramètre de type sans contrainte de classe

'Is' operand of type '<typeparametername>' can be compared only to 'Nothing' because '<typeparametername>' is a type parameter with no class constraint

Un paramètre de type est utilisé comme opérande pour Is, opérateur (Visual Basic) lorsqu'il est défini sans le mot clé Class (Visual Basic) ou sans un nom de classe spécifique dans sa liste de contraintes.

Is compare deux types référence pour déterminer s'ils pointent vers la même instance d'objet dans la mémoire. Il ne peut pas accepter un opérande qui n'est pas un type référence à moins que l'autre opérande soit Nothing (Visual Basic).

ID d'erreur : BC32052

Pour corriger cette erreur

- Si vous souhaitez que l'argument de type fourni à ce paramètre de type soit toujours un type référence, ajoutez le mot clé **Class** ou un nom de classe spécifique à la liste des contraintes pour le paramètre de type.
- Si vous ne souhaitez pas que l'argument de type fourni à ce paramètre de type soit toujours un type référence, supprimez-le de l'expression **Is**. Vous ne pouvez pas le comparer à d'autres types référence à l'aide de l'opérateur **Is**.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic
Types valeur et types référence
Opérateurs de comparaison en Visual Basic

'Is' attendu

'Is' expected

Un opérateur **TypeOf** apparaît sans clause **Is**.

ID d'erreur : BC30224

Pour corriger cette erreur

- Ajoutez une clause **Is** à la suite de l'opérateur **TypeOf** ; par exemple `TypeOf MyControl Is ComboBox`.

Voir aussi

Concepts

Opérateurs de comparaison en Visual Basic

Résolution à liaison tardive ; des erreurs d'exécution peuvent se produire

Late bound resolution; runtime errors could occur

Un objet est assigné à une variable déclarée comme étant du type de données Object.

Lorsque vous déclarez une variable comme **Object**, le compilateur doit exécuter une *liaison tardive* qui provoque des opérations supplémentaires au moment de l'exécution. Il expose également votre application à d'éventuelles erreurs d'exécution. Par exemple, si vous assignez Form à la variable **Object**, puis vous essayez d'accéder à la propriété System.Xml.XmlDocument.NameTable, le runtime lève une exception MemberAccessException car la classe **Form** n'expose pas de propriété `NameTable`.

Si vous déclarez la variable comme étant d'un type spécifique, le compilateur peut exécuter une *liaison anticipée* au moment de la compilation. Cela entraîne des performances améliorées, un accès contrôlé aux membres du type spécifique et une meilleure lisibilité de votre code.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42017

Pour corriger cette erreur

- Si possible, déclarez la variable comme étant d'un type spécifique.

Voir aussi

Concepts

Liaison anticipée et liaison tardive
Déclaration des variables objets

Les étiquettes qui sont des nombres doivent être suivies du signe deux-points

Labels that are numbers must be followed by colons

Les numéros de ligne sont soumis aux même règles que les autres types d'étiquettes et doivent contenir le signe deux-points.

ID d'erreur : BC30801

Pour corriger cette erreur

- Placez le numéro suivi de deux-points au début d'une ligne de code. Par exemple :

```
400:    X += 1
```

Voir aussi

Référence

GoTo, instruction

Les étiquettes ne sont pas valides en dehors des méthodes

Labels are not valid outside methods

Vous pouvez ajouter une étiquette à une instruction uniquement à l'intérieur de **Sub**, de **Function**, d'une propriété **Get** ou d'une procédure de propriété **Set**. Seule une instruction exécutable peut avoir une étiquette et toutes les instructions exécutables doivent se trouver à l'intérieur de procédures.

ID d'erreur : BC30016

Pour corriger cette erreur

- Supprimez l'étiquette de l'instruction ou déplacez l'instruction à l'intérieur d'une procédure.

Voir aussi

Tâches

Comment : étiqueter des instructions

Référence

Sub, instruction (Visual Basic)
Function, instruction (Visual Basic)
Get, instruction
Set, instruction (Visual Basic)

Les étiquettes ne sont pas valides dans la fenêtre Exécution

Labels are not valid in the Immediate window

Les étiquettes de ligne permettent le branchement et ne sont pas autorisées dans un contexte de débogage.

ID d'erreur : BC30134

Pour corriger cette erreur

- N'entrez pas d'étiquette de ligne dans la fenêtre **Exécution**.

Voir aussi

Tâches

Comment : étiqueter des instructions

Référence

Fenêtre Exécution

L'étiquette '<nométiquette>' n'est pas définie

Label '<labelname>' is not defined

Il est fait référence à une étiquette de ligne ou à un numéro de ligne qui n'est pas défini dans la portée de la référence. L'étiquette doit se trouver à l'intérieur de la procédure qui contient la référence.

ID d'erreur : BC30132

Pour corriger cette erreur

- Restructurez le code de manière à ce que l'étiquette de ligne et la référence se trouvent dans la même procédure.

Voir aussi

Tâches

Comment : étiqueter des instructions

Référence

Property, instruction

Concepts

Procédures dans Visual Basic

L'étiquette '<nométiquette>' est déjà définie dans la méthode actuelle

Label '<labelname>' is already defined in the current method

Le nom d'étiquette est dupliqué à l'intérieur de la procédure. Tous les noms doivent être uniques au sein d'une procédure.

ID d'erreur : BC30094

Pour corriger cette erreur

- Renommez l'étiquette dupliquée.

Voir aussi

Tâches

Comment : étiqueter des instructions

Concepts

Noms d'éléments déclarés

Mot clé non valide en tant qu'identificateur

Keyword is not valid as an identifier

Un mot clé apparaît lorsqu'un nom d'élément est requis. Le contexte nécessite un élément tel qu'une variable ou une procédure.

ID d'erreur : BC30183

Pour corriger cette erreur

- Remplacez le mot clé par un nom d'élément valide.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

Le mot clé ne désigne pas un type

Keyword does not name a type

Un mot clé apparaît là où un nom de type est requis. Le contexte nécessite soit un type de données élémentaire, soit un nom d'élément déclaré représentant un type.

ID d'erreur : BC30180

Pour corriger cette erreur

- Remplacez le mot clé par un nom de type valide.

Voir aussi

Référence

Class, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Function, instruction (Visual Basic)
Property, instruction
Structure, instruction

Un caractère '.' ou '!' de début ne peut apparaître que dans une instruction 'With'

Leading '.' or '!' can only appear inside a 'With' statement

Un point (.) ou un point d'exclamation (!) qui ne se trouve pas à l'intérieur d'un bloc **With** apparaît sans expression à gauche. L'accès au membre (**.**) et l'accès au membre dictionnaire (**!**) requièrent une expression spécifiant l'élément contenant le membre. Celui-ci doit apparaître immédiatement à gauche de l'accesseur ou comme cible d'un bloc **With** contenant l'accès au membre.

ID d'erreur : BC30157

Pour corriger cette erreur

1. Vérifiez que le bloc **With** est correctement mis en forme.
2. En l'absence de bloc **With**, ajoutez une expression à gauche de l'accesseur qui correspond à un élément défini contenant le membre.

Voir aussi

Référence

With...End With, instruction (Visual Basic)

Concepts

Caractères spéciaux dans le code

La résolution de surcharge à liaison tardive ne peut pas être appliquée à '<NomProcédure>', car l'instance d'accès est un type interface

Latebound overload resolution cannot be applied to '<procedurename>' because the accessing instance is an interface type

Le compilateur essaie de résoudre une référence à une propriété ou une procédure surchargée, mais la référence échoue parce qu'un argument est de type **Object** et l'objet de référence a le type de données d'une interface. L'argument **Object** oblige le compilateur à résoudre la référence en tant que liaison tardive.

Dans ces cas, le compilateur résout la surcharge par l'intermédiaire de la classe d'implémentation au lieu de l'interface sous-jacente. Si la classe renomme l'une des versions surchargées, le compilateur ne considère pas cette version comme une surcharge car son nom est différent. Il ignore la version renommée alors qu'elle représente peut-être le choix correct pour résoudre la référence.

ID d'erreur : BC30933

Pour corriger cette erreur

- Utilisez **CType** pour effectuer un cast de l'argument de **Object** dans le type spécifié par la signature de la surcharge que vous souhaitez appeler.

  Notez qu'il ne permet pas d'effectuer un cast de l'objet de référence dans l'interface sous-jacente. Vous devez effectuer un cast de l'argument pour éviter cette erreur.

Exemple

L'exemple suivant présente un appel à une procédure **Sub** surchargée qui génère cette erreur au moment de la compilation.

```
Module m1
    Interface i1
        Sub s1(ByVal p1 As Integer)
        Sub s1(ByVal p1 As Double)
    End Interface
    Class c1
        Implements i1
        Public Overloads Sub s1(ByVal p1 As Integer) Implements i1.s1
        End Sub
        Public Overloads Sub s2(ByVal p1 As Double) Implements i1.s1
        End Sub
    End Class
    Sub Main()
        Dim refer As i1 = New c1
        Dim o1 As Object = 3.1415
        ' The following reference is INVALID and causes a compiler error.
        refer.s1(o1)
    End Sub
End Module
```

Dans l'exemple précédent, si le compilateur a autorisé l'appel à `s1` tel qu'il est écrit, la résolution a lieu par l'intermédiaire de la classe `c1` au lieu de l'interface `i1`. Cela signifie que le compilateur ne tient pas compte de `s2` car son nom est différent dans `c1`, même s'il représente le choix correct, tel que défini par `i1`.

Vous pouvez corriger l'erreur en modifiant l'appel à l'une ou l'autre des lignes suivantes du code :

```
refer.s1(CType(o1, Integer))
refer.s1(CType(o1, Double))
```

Chacune des lignes précédentes du code effectue un cast explicite de la variable **Object** `o1` dans l'un des types de paramètre définis pour les surcharges.

Voir aussi

Référence

CType, fonction

Concepts

Surcharge de procédure

Résolution de surcharge

'Lib' attendu

'Lib' expected

Une instruction **Declare** spécifie un nom de bibliothèque sans utiliser le mot clé **Lib**.

ID d'erreur : BC30218

Pour corriger cette erreur

- Ajoutez le mot clé **Lib** avant le nom de la bibliothèque.

Voir aussi

Référence

Declare, instruction

Les instructions d'assignation 'Let' et 'Set' ne sont plus prises en charge

'Let' and 'Set' assignment statements are no longer supported

Vous ne pouvez pas utiliser **Let** et **Set** dans des assignations.

ID d'erreur : BC30807

Pour corriger cette erreur

- Omettez **Let** et **Set** des instructions d'assignation. Par exemple :

```
X = 2
```

Voir aussi

Référence

Property, instruction

Concepts

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

Les variables locales à l'intérieur des méthodes des structures ne peuvent pas être déclarées 'Static'

Local variables within methods of structures cannot be declared 'Static'

Vous ne pouvez pas utiliser le modificateur **Static** avec des variables locales dans des structures.

ID d'erreur : BC31400

Pour corriger cette erreur

1. Supprimez le modificateur **Static** de la variable locale.
2. Déclarez la variable en tant que variable statique avec une portée plus large.

Voir aussi

Référence

Static (Visual Basic)

Les variables locales dans les méthodes génériques ne peuvent pas être déclarées 'Static'

Local variables within generic methods cannot be declared 'Static'

La déclaration d'une variable locale dans une procédure générique spécifie **Static**.

À l'heure actuelle, Visual Basic et .NET Framework ne prennent en charge aucune combinaison de variables statiques et de procédures génériques.

ID d'erreur : BC32068

Pour corriger cette erreur

- Supprimez le mot clé **Static** de la déclaration de variable.

Voir aussi

Tâches

Comment : augmenter la durée de vie d'une variable

Référence

Static (Visual Basic)

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Une variable locale peut ne pas avoir le même nom que la fonction qui la contient

Local variable cannot have the same name as the function containing it

Une instruction **Dim** spécifie une variable avec le même nom que la procédure **Function** qui la contient.

ID d'erreur : BC30290

Pour corriger cette erreur

- Supprimez la déclaration de variable ou changez le nom de la variable.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom
Function, procédures

La variable locale '<nomvariable>' est déjà déclarée dans le bloc en cours

Local variable '<variablename>' is already declared in the current block

Deux variables de même nom sont déclarées dans le même bloc (qui peut être une procédure ou un bloc de contrôle, tel que **If...Then**).

ID d'erreur : BC30288

Pour corriger cette erreur

- Déplacez une des déclarations vers un bloc différent ou modifiez le nom de l'une des variables.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom
Occultation dans Visual Basic

Impossible de faire référence à la variable locale '<nomvariable>' avant qu'elle ne soit déclarée

Local variable '<variablename>' cannot be referred to before it is declared

À l'intérieur d'une procédure, une variable est utilisée avant sa déclaration.

ID d'erreur : BC32000

Pour corriger cette erreur

- Déplacez la déclaration de sorte qu'elle précède la première référence à la variable.

Voir aussi

Concepts

Déclaration de variable dans Visual Basic

Les instructions 'Line' ne sont plus prises en charge (erreur du compilateur Visual Basic)

'Line' statements are no longer supported (Visual Basic Compiler Error)

Les instructions Line ne sont plus prises en charge. Les fonctionnalités d'E/S sur fichier sont disponibles en tant que **Microsoft.VisualBasic.FileSystem.LineInput** et les fonctionnalités graphiques en tant que **System.Drawing.Graphics.DrawLine**.

ID d'erreur : BC30830

Pour corriger cette erreur

1. Pour accéder au fichier, utilisez **Microsoft.VisualBasic.FileSystem.LineInput**.
2. Pour exécuter des graphiques, utilisez **System.Drawing.Graphics.Drawline**.

Voir aussi

Référence

System.IO Namespace

System.Drawing Namespace

Autres ressources

Accès au fichier avec Visual Basic

Les instructions 'Line' ne sont plus prises en charge (erreur du compilateur Visual Basic/Smart Device)

'Line' statements are no longer supported (Smart Device/Visual Basic Compiler Error)

L'instruction **Line** n'est plus prise en charge. La fonctionnalité d'E/S de fichier est normalement disponible en tant que Microsoft.VisualBasic.FileSystem.LineInput(System.Int32), mais la version ciblée du .NET Compact Framework ne la prend pas en charge.

ID d'erreur : BC30768

Pour corriger cette erreur

- Pour accéder au fichier, utilisez les fonctions définies dans l'espace de noms System.IO.
- Pour exécuter des graphiques, utilisez System.Drawing.Graphics.DrawLine.

Voir aussi

Référence

System.IO

System.Drawing

Autres ressources

Accès au fichier avec Visual Basic

Ligne trop longue

Line is too long

Les lignes de texte sources ne peuvent pas comporter plus de 65 535 caractères.

ID d'erreur : BC30494

Pour corriger cette erreur

- Raccourcissez la longueur de la ligne à 65 535 caractères ou moins.

Voir aussi

Concepts

Types d'erreurs

Une variable de contrôle de boucle ne peut pas être une propriété ni un tableau indexé à liaison tardive

Loop control variable cannot be a property or a late-bound indexed array

La variable utilisée pour effectuer une itération dans une boucle **For** doit présenter un type de données numérique.

ID d'erreur : BC30039

Pour corriger cette erreur

- Modifiez la déclaration de la variable de contrôle de boucle pour spécifier **Integer**, **Short**, **Long**, **Byte**, **Single**, **Double** ou **Decimal**.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

'Loop' ne peut pas avoir de condition si le 'Do' correspondant en a une

'Loop' cannot have a condition if matching 'Do' has one

Une instruction **Loop** contient une clause **While** ou **Until** et l'instruction **Do** correspondante contient également cette clause. Une seule des deux instructions **Do** et **Loop** d'une boucle peut spécifier une condition.

ID d'erreur : BC30238

Pour corriger cette erreur

- Supprimez la clause **While** ou **Until** de l'instruction **Do** ou de l'instruction **Loop**.

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)

Les instructions de boucle ne sont pas valides dans la fenêtre Exécution

Loop statements are not valid in the Immediate window

Vous ne pouvez pas utiliser les instructions **Loop** dans la fenêtre **Exécution**.

ID d'erreur : BC30708

Pour corriger cette erreur

- Supprimez les instructions **Loop** du code de débogage.

Voir aussi

Référence

Loop

Autres ressources

Débogage dans Visual Studio

'Loop' doit être précédé d'un 'Do' correspondant

'Loop' must be preceded by a matching 'Do'

Une instruction **Loop** apparaît sans une instruction **Do** correspondante. Une instruction **Loop** doit être précédée d'une instruction **Do** correspondante.

ID d'erreur : BC30091

Pour corriger cette erreur

1. Si cette boucle **Do** fait partie d'un ensemble de boucles **Do** imbriquées, vérifiez que chaque boucle est correctement terminée.
2. Vérifiez que les autres structures de contrôle à l'intérieur de la boucle **Do** sont correctement terminées.
3. Vérifiez que cette boucle **Do** est correctement mise en forme.

Voir aussi

Référence

Do...Loop, instruction (Visual Basic)

Les membres d'une structure ne peuvent pas être déclarés '<spécificateur>'

Members in a Structure cannot be declared '<specifier>'

Vous avez utilisé un spécificateur non valide pour un membre dans une instruction **Structure**.

ID d'erreur : BC30435

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Concepts

Structures et classes

Autres ressources

Structures constituées à partir de vos propres types de données

Les membres dans un module ne peuvent pas être déclarés '<spécificateur>'

Members in a Module cannot be declared '<specifier>'

Vous avez utilisé un spécificateur non valide pour un membre dans une instruction **Module**. Les modules ne peuvent jamais être instanciés, ne prennent pas en charge l'héritage et ne peuvent pas implémenter des interfaces.

ID d'erreur : BC30436

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Module, instruction

Le membre '<NomMembre1>' déclare implicitement '<NomMembreImplicite>', qui est en conflit avec un membre dans la classe de base '<NomClasseDeBase>'

Message d'erreur
Le membre '<NomMembre1>' déclare implicitement '<NomMembreImplicite>', qui est en conflit avec un membre dans la classe de base '<NomClasseDeBase>'. Par conséquent, le membre ne doit pas être déclaré 'Overloads'
Member '<membername1>' implicitly declares '<implicitmembername>', which conflicts with a member in the base class '<baseclassname>', and so the member should not be declared 'Overloads'

Une propriété d'une classe dérivée génère un membre implicite du même nom qu'un membre de la classe de base et spécifie le mot clé Overloads.

La surcharge permet de définir de nombreuses versions d'une propriété ou d'une procédure dans la même classe. Vous ne pouvez pas définir une version supplémentaire d'un membre de classe de base tant que ce membre ne spécifie pas **Overloads**. Étant donné que le membre de la classe de base en conflit ne spécifie pas **Overloads**, le compilateur suppose que cette propriété Shadows utilise le membre de la classe de base implicite.

Le compilateur Visual Basic crée des membres implicites correspondant à certains éléments de programmation que vous déclarez. Le tableau suivant récapitule ces membres implicit ou *synthetic*.

Élément déclaré	**Membres créés implicitement**
Enumération	Membre `value__`
Événement	Procédure `add_<eventname>` Procédure `remove_<eventname>` Champ `<eventname>Event` Délégué `<eventname>EventHandler`
Propriété	Procédure `get_<propertyname>` Procédure `set_<propertyname>`
Membre de **My.Form**, de **My.WebService** ou d'une classe marquée avec l'attribut MyGroupCollectionAttribute	`m_<variablename>` Variable **Static** Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`
Variable **WithEvents**	Variable `_<variablename>` Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`

En raison de la possibilité de conflits de noms, vous devez éviter de nommer tout élément de programmation déclaré à l'aide

du même formulaire que l'un de ces membres implicites. Vous devez par exemple éviter les noms d'éléments commençant par `get_` ou `set_`.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40022

Pour corriger cette erreur

- Si vous avez l'intention de masquer ou d'occulter le membre de la classe de base, remplacez le mot clé Overloads par le mot clé Shadows dans la déclaration de la propriété.
- Si vous n'avez pas l'intention d'occulter le membre de la classe de base, modifiez le nom de la propriété pour éviter les conflits de noms décrits dans le tableau précédent.

Voir aussi

Concepts

Noms d'éléments déclarés

Le membre '<NomMembre1>' déclare implicitement '<NomMembreImplicite>', qui est en conflit avec un membre implicitement déclaré pour le membre '<NomMembre2>' dans la classe de base '<NomClasseDeBase>'

Message d'erreur
Le membre '<NomMembre1>' déclare implicitement '<NomMembreImplicite>', qui est en conflit avec un membre implicitement déclaré pour le membre '<NomMembre2>' dans la classe '<NomClasseDeBase>' de base. Le membre doit donc être déclaré 'Shadows'.
Member '<membername1>' implicitly declares '<implicitmembername>', which conflicts with a member implicitly declared for member '<membername2>' in the base class '<baseclassname>'. So the member should be declared 'Shadows'.

Un membre d'une classe dérivée génère un membre implicite qui porte le même nom qu'un membre implicite de la classe de base. Étant donné que les membres implicites ne spécifient pas Overloads, le compilateur suppose que ce membre Shadows le membre implicite de la classe de base. Votre code est plus lisible et moins susceptible d'engendrer des erreurs, si vous spécifiez explicitement le mot clé **Shadows** pour ce membre.

Le compilateur Visual Basic crée des membres implicites correspondant à certains éléments de programmation que vous déclarez. Le tableau suivant récapitule ces membres implicit ou *synthetic*.

Élément déclaré	**Membres créés implicitement**
Énumération	Membre `value__`
Événement	Procédure `add_<eventname>` Procédure `remove_<eventname>` Champ `<eventname>Event` Délégué `<eventname>EventHandler`
Propriété	Procédure `get_<propertyname>` Procédure `set_<propertyname>`
Membre de **My.Form**, de **My.WebService** ou d'une classe marquée avec l'attribut MyGroupCollectionAttribute	`m_<variablename>` Variable **Static** Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`
Variable **WithEvents**	Variable `_<variablename>` Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`

En raison de la possibilité de conflits de noms, vous devez éviter de nommer tout élément de programmation déclaré à l'aide

du même formulaire que l'un de ces membres implicites. Vous devez par exemple éviter les noms d'éléments commençant par `get_` ou `set_`.

Ce message est un avertissement par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40018

Pour corriger cette erreur

- Si vous comptez masquer ou occulter le membre implicite de la classe de base, incluez le mot clé Shadows dans la déclaration du membre de la classe dérivée.
- Si vous ne comptez pas occulter le membre implicite de la classe de base, modifiez le nom du membre de la classe dérivée pour éviter tout conflit avec les noms répertoriés dans le précédent tableau.

Voir aussi

Concepts

Noms d'éléments déclarés

Le membre '<NommMembre1>' est en conflit avec un membre déclaré implicitement pour le membre '<NomMembre2>' dans le type de base '<NomTypeBase>' et ne devrait donc pas être déclaré 'Overloads'

Member '<membername1>' conflicts with a member implicitly declared for member '<membername2>' in the base type '<basetypename>' and so should not be declared 'Overloads'

Une propriété ou une procédure dans une classe dérivée utilise le même nom qu'un membre implicite de la classe de base et spécifie le mot clé Overloads.

La surcharge est utilisée pour définir plusieurs versions d'une propriété ou d'une procédure dans la même classe. Vous ne pouvez pas définir une version supplémentaire d'un membre de la classe de base à moins que celui-ci spécifie déjà **Overloads**. Étant donné que les membres implicites ne spécifient pas **Overloads**, le compilateur suppose que cette propriété ou procédure Shadows le membre implicite de la classe de base.

Le compilateur Visual Basic crée des membres implicites correspondant à certains éléments de programmation que vous déclarez. Le tableau suivant récapitule ces membres implicit ou *synthetic*.

Élément déclaré	**Membres créés implicitement**
Énumération	Membre `value__`
Événement	Procédure `add_<eventname>` Procédure `remove_<eventname>` Champ `<eventname>Event` Délégué `<eventname>EventHandler`
Propriété	Procédure `get_<propertyname>` Procédure `set_<propertyname>`
Membre de **My.Form**, de **My.WebService** ou d'une classe marquée avec l'attribut MyGroupCollectionAttribute	`m_<variablename>` Variable **Static** Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`
Variable **WithEvents**	Variable `_<variablename>` Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`

En raison de la possibilité de conflits de noms, vous devez éviter de nommer tout élément de programmation déclaré à l'aide du même formulaire que l'un de ces membres implicites. Vous devez par exemple éviter les noms d'éléments commençant par

`get_` ou `set_`.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements et le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40023

Pour corriger cette erreur

- Modifiez le nom de la propriété ou la procédure pour éviter tout conflit avec les noms répertoriés dans le précédent tableau.

Voir aussi

Concepts

Noms d'éléments déclarés

Le membre '<NomMembre>' définit implicitement un membre '<NomMembreImplicite>' qui porte le même nom qu'un paramètre de type

Member '<membername>' implicitly defines a member '<implicitmembername>' which has the same name as a type parameter

Un membre d'une classe générique génère un membre implicite avec le même nom qu'un paramètre de type de la classe.

Le compilateur Visual Basic crée des membres implicites correspondant à certains éléments de programmation que vous déclarez. Le tableau suivant récapitule ces membres implicit ou *synthetic*.

Élément déclaré	**Membres créés implicitement**
Enumération	Membre `value__`
Événement	Procédure `add_<eventname>` Procédure `remove_<eventname>` Champ `<eventname>Event` Délégué `<eventname>EventHandler`
Propriété	Procédure `get_<propertyname>` Procédure `set_<propertyname>`
Variable de collection **My.**	`m_<variablename>` Variable **Static** Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`
Variable **WithEvents**	Variable `_<variablename>` Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`

En raison de la possibilité de conflits de noms, vous devez éviter de nommer tout élément de programmation déclaré à l'aide du même formulaire que l'un de ces membres implicites. Vous devez par exemple éviter les noms d'éléments commençant par `get_` ou `set_`.

ID d'erreur : BC32070

Pour corriger cette erreur

- Si le nom du paramètre de type est flexible, modifiez-le pour éviter des conflits avec les noms répertoriés dans le tableau précédent.
- Si le paramètre de type doit conserver son nom, modifiez le nom du membre de classe pour éviter des conflits avec les noms répertoriés dans le tableau précédent.

Voir aussi

Référence

Liste de types

Concepts

Noms d'éléments déclarés

Le membre '<NomMembre>' est en conflit avec le membre '<NomMembre>' dans le type de base '<NomTypeBase>' et ne devrait donc pas être déclaré 'Overloads'

Member '<membername>' conflicts with member '<membername>' in the base type '<basetypename>' and so should not be declared 'Overloads'

Une propriété ou une procédure utilise le mot clé Overloads pour redéclarer une propriété ou une procédure existante avec le même nom, mais cette propriété ou procédure existante figure dans la classe de base.

La surcharge est utilisée pour définir plusieurs versions d'une propriété ou d'une procédure dans la même classe. Vous ne pouvez pas définir une version supplémentaire d'un membre de la classe de base à moins que celui-ci spécifie déjà Overloads.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40021

Pour corriger cette erreur

- Si vous comptez définir une version supplémentaire du membre de la classe de base et si vous avez accès au code source de la classe de base, ajoutez le mot clé Overloads à la définition de la classe de base.
- Si vous n'avez pas accès au code source de la classe de base, vous ne pouvez pas surcharger le membre d'une classe dérivée. Supprimez le mot clé **Overloads**.
- Si vous souhaitez remplacer le membre de la classe de base au lieu de définir une version supplémentaire de celui-ci, utilisez le mot clé Overrides au lieu de **Overloads**.
- Si vous souhaitez masquer le membre de la classe de base avec un nouveau membre de la classe dérivée, utilisez le mot clé Shadows au lieu de **Overloads**.

Voir aussi

Concepts

Surcharge de procédure
Éléments fondamentaux de l'héritage

Le membre '<nommembre>' est introuvable dans la classe '<nomclasse>'

Message d'erreur
Le membre '<nommembre>' est introuvable dans la classe '<nomclasse>'. Cette condition résulte généralement d'un fichier 'Microsoft.VisualBasic.dll' incompatible.
Member '<membername>' cannot be found in class '<classname>'. This condition is usually the result of a mismatched 'Microsoft.VisualBasic.dll'.

Un membre défini n'a pas pu être trouvé.

ID d'erreur : BC31097

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si l'erreur se reproduit, enregistrez votre travail et redémarrez Visual Studio.
3. Si l'erreur persiste, réinstallez Visual Basic.
4. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources
Support technique et accessibilité

Le membre '<NomClasse>.<NomProcédure>' qui correspond à cette signature ne peut pas être substitué, car la classe '<NomClasse>' contient plusieurs membres avec le même nom et la même signature : <ListeSignatures>

Member '<classname>.<procedurename>' that matches this signature cannot be overridden because the class '<classname>' contains multiple members with this same name and signature: <signaturelist>

Une procédure ou une propriété tente de substituer une procédure ou une propriété héritée, mais le compilateur détecte plusieurs versions de la procédure ou de la propriété de base contenant le même nom et la même signature.

Cette erreur peut se produire dans le cas de types génériques construits, comme l'illustre le squelette des déclarations ci-après.

```
Public Class baseClass(Of t)
    Public Overridable Sub doSomething(ByVal inputValue As String)
    End Sub
    Public Overridable Sub doSomething(ByVal inputValue As t)
    End Sub
End Class
Public Class derivedClass
    Inherits baseClass(Of String)
    Overrides Sub doSomething(ByVal inputValue As String)
    End Sub
End Class
```

Étant donné que `derivedClass` hérite de `baseClass` qui fournit **String** à son paramètre de type `t`, les deux versions de `doSomething` dans `baseClass` acceptent des signatures identiques pour `derivedClass`. En conséquence, le compilateur ne peut pas déterminer la version à substituer.

ID d'erreur : BC30935

Pour corriger cette erreur

- Modifiez le ou les arguments de type fournis à la classe de base afin qu'il n'existe pas une ou plusieurs signatures identiques des procédures ou des propriétés membres.

  - ou -

- Si vous devez hériter de la classe de base le jeu de types d'arguments que vous utilisez, ne substituez pas la procédure ou la propriété mentionnée dans ce message d'erreur.

Voir aussi

Référence

Overridable

Overrides

Concepts

Substitution de propriétés et de méthodes

Le membre '<NomInterface>.<NomProcédure>' qui correspond à cette signature ne peut pas être implémenté, car l'interface '<NomInterface>' contient plusieurs membres avec le même nom et la même signature : <ListeSignatures>

Member '<interfacename>.<procedurename>' that matches this signature cannot be implemented because the interface '<interfacename>' contains multiple members with this same name and signature: <signaturelist>

Une procédure ou une propriété essaie d'en implémenter une autre définie dans une interface implémentée, mais le compilateur rencontre plusieurs versions de la procédure ou de la propriété de l'interface portant les mêmes nom et signature.

Cette erreur peut se produire dans une situation avec les types génériques construits, comme le montrent les déclarations de squelette suivantes.

```
Public Interface baseInterface(Of t)
    Sub doSomething(ByVal inputValue As String)
    Sub doSomething(ByVal inputValue As t)
End Class
Public Class implementingClass
    Implements baseInterface(Of String)
    Sub doSomething(ByVal inputValue As String) _
        Implements baseInterface(Of String).doSomething
    End Sub
End Class
```

Étant donné que `implementingClass` implémente `baseInterface` qui fournit **String** à son paramètre de type `t`, les deux versions de `doSomething` dans `baseInterface` prennent des signatures identiques en ce qui concerne `implementingClass`. Par conséquent, le compilateur ne peut pas déterminer quelle version implémenter.

ID d'erreur : BC30937

Pour corriger cette erreur

- Modifiez l'argument de type ou les arguments que vous fournissez à la classe de base de sorte qu'ils ne produisent pas une ou plusieurs signatures identiques de procédures ou de propriétés de membres.

  - ou -

- N'implémentez pas cette classe de base. Vous ne pouvez pas l'implémenter avec l'ensemble des arguments de type que vous utilisez, parce que vous devez implémenter chacun de ses membres.

Voir aussi

Référence

Implements (Visual Basic)
Implements, instruction

Concepts

Implements, mot clé et instruction

'Me' ne peut pas être la cible d'une assignation

'Me' cannot be the target of an assignment

Le mot clé **Me** fait toujours référence à l'instance spécifique d'une classe dans laquelle le code est en train de s'exécuter. Il n'est pas possible de changer sa valeur.

ID d'erreur : BC30062

Pour corriger cette erreur

- Supprimez l'instruction qui assigne une valeur à **Me**.

Voir aussi

Référence

Me

Concepts

Assignation des variables objets

Le nombre maximal d'erreurs est dépassé

Maximum number of errors has been exceeded

Le nombre maximal d'erreurs de compilation s'élève à 100. Le compilateur impose cette limite afin de réduire la taille des rapports d'erreurs.

ID d'erreur : BC30041

Pour corriger cette erreur

- Examinez les autres erreurs générées par cette compilation et résolvez-en le plus possible. Si le code source contient plus de 100 erreurs, davantage d'erreurs pourront apparaître dans la compilation suivante.

Voir aussi

Concepts

Configuration d'avertissements en Visual Basic

Autres ressources

Compilateur Visual Basic

Le nombre maximal d'avertissements est dépassé

Maximum number of warnings has been exceeded

Le nombre maximal d'avertissements de compilation s'élève à 100. Le compilateur impose cette limite afin de réduire la taille des rapports d'erreurs et d'avertissements.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42206

Pour corriger cette erreur

- Examinez les autres avertissements générés par cette compilation et résolvez-en le plus possible. Si le code source contient plus de 100 avertissements, davantage d'avertissements pourront apparaître dans la compilation suivante.

Voir aussi

Concepts

Configuration d'avertissements en Visual Basic

Autres ressources

Compilateur Visual Basic

Un opérateur '<SymboleOpérateur1>' correspondant est requis pour '<SymboleOpérateur2>'

Matching '<operatorsymbol1>' operator is required for '<operatorsymbol2>'

Un opérateur est défini alors que son opérateur correspondant requis n'est pas défini.

Les opérateurs suivants doivent être définis comme des paires correspondantes :

- **=** et **<>**
- **>** et **<**
- **>=** et **<=**
- **IsTrue** et **IsFalse**

Si vous définissez chacun de ces opérateurs dans une classe ou dans une structure, vous devez également définir leurs opérateurs correspondants dans la même classe ou structure.

Même si vous n'utilisez pas explicitement l'opérateur correspondant, Visual Basic peut en avoir besoin. Par exemple, si vous définissez une classe ou une structure que vous utilisez comme variable compteur dans un For...Next, instruction (Visual Basic), Visual Basic requiert les opérateurs **>=** et **<=** (ainsi que l'opérateur **+**).

ID d'erreur : BC33033

Pour corriger cette erreur

- Définissez l'opérateur correspondant dans la même classe ou structure. Veillez à lui donner une définition significative, car Visual Basic peut l'utiliser dans une situation que vous ne prévoyez pas.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Opérateurs et expressions en Visual Basic

La méthode '<nomméthode>' dispose d'une demande de liaison, mais se substitue à ou implémente les méthodes suivantes qui n'ont pas de demande de liaison. Un risque potentiel de sécurité existe :

Method '<methodname>' has a link demand, but overrides or implements the following methods which do not have a link demand. A security hole may exist:

Une action de demande de liaison de sécurité a été ajoutée à la méthode. Cependant, la méthode se substitue aux méthodes ne disposant pas de demande de liaison ou les implémente. Les méthodes substituées ou implémentées peuvent donc être appelées avec des autorisations insuffisantes, ce qui entraîne des problèmes de sécurité.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42200

Pour corriger cette erreur

- Ajoutez des demandes de liaison aux méthodes substituées et/ou implémentées.

Voir aussi

Concepts

Demandes de liaison
Demand et LinkDemand
Optimisations de la sécurité

La méthode '<nomméthode>' n'a pas la même signature que le délégué '<nomdélégué>'

Method '<methodname>' does not have the same signature as delegate '<delegatename>'

Il n'y a pas de correspondance entre les signatures de la méthode et le délégué que vous tentez d'utiliser. L'instruction **Delegate** définit les types de paramètres et les types de retour d'une classe déléguée. Les procédures présentant des paramètres, types et types de retour assortis peuvent être utilisées pour créer une instance de ce type délégué. Chaque classe déléguée définit un constructeur auquel les caractéristiques d'une méthode objet sont passées.

ID d'erreur : BC30408

Pour corriger cette erreur

- Vérifiez les signatures pour rechercher l'incompatibilité et corrigez celle-ci.

Voir aussi

Référence

Delegate, instruction

La méthode '<nomméthode>' ne peut pas gérer l'événement '<nomévénement>', car ils n'ont pas la même signature

Method '<methodname>' cannot handle Event '<eventname>' because they do not have the same signature

Les paramètres ou types de paramètres du gestionnaire d'événements spécifié ne correspondent pas à ceux définis pour l'événement spécifié.

ID d'erreur : BC31029

Pour corriger cette erreur

- Redéfinissez l'événement ou le gestionnaire d'événements de manière à faire correspondre les listes de paramètres.

Voir aussi

Autres ressources

Événements en Visual Basic

Une méthode dans une structure ne peut pas être déclarée 'Protected' ni 'Protected Friend'

Method in a structure cannot be declared 'Protected' or 'Protected Friend'

Une méthode dans une structure a été déclarée en tant que **Protected** ou **Protected Friend**.

ID d'erreur : BC31067

Pour corriger cette erreur

- Déclarez les méthodes à l'intérieur des structures en tant que **Public** ou **Private**.

Voir aussi

Référence

Structure, instruction
Protected (Visual Basic)
Friend (Visual Basic)

Une instruction de déclaration de méthode doit être la première instruction d'une ligne logique

Method declaration statements must be the first on a logical line

Une instruction **Function** ou **Sub** suit un séparateur d'instruction deux-points (:). **Function** ou **Sub** doit être la seule instruction sur sa ligne source.

ID d'erreur : BC32009

Pour corriger cette erreur

- Séparez les instructions multiples en les disposant sur différentes lignes.

Voir aussi

Tâches

Comment : diviser et combiner des instructions dans le code

Référence

Sub, instruction (Visual Basic)

Function, instruction (Visual Basic)

La méthode ne peut pas comporter simultanément des paramètres ParamArray et Optional

Method cannot have both a ParamArray and Optional parameters

Un argument **ParamArray** est automatiquement facultatif et il doit être le seul argument facultatif de la déclaration de procédure. Tous les arguments précédents doivent être requis.

ID d'erreur : BC30046

Pour corriger cette erreur

- Supprimez les arguments facultatifs dans la déclaration.

Voir aussi

Référence

ParamArray

Concepts

Tableaux de paramètres

Une méthode ne peut pas contenir à la fois une instruction 'Try' et une instruction 'On Error' ou 'Resume'

Method cannot contain both a 'Try' statement and an 'On Error' or 'Resume' statement

Vous avez associé une instruction **Try** à une instruction **On Error** ou **Resume**.

ID d'erreur : BC30544

Pour corriger cette erreur

- Supprimez l'instruction **On Error** ou **Resume**.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

L'appel de méthode n'a retourné aucune valeur

Method call did not return a value

La méthode appelée n'a pas retourné de valeur comme prévu.

ID d'erreur : BC30723

Pour corriger cette erreur

- Vérifiez que la méthode appelée est une fonction qui retourne une valeur.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Les arguments de méthode doivent être mis entre parenthèses

Method arguments must be enclosed in parentheses

Les règles qui régissent les appels de procédure sont plus simples dans les dernières versions de Visual Basic. Tous les arguments doivent être mis entre parenthèses.

ID d'erreur : BC30800

Pour corriger cette erreur

- Mettez la liste d'arguments entre parenthèses. Par exemple :

```
MySub(ArrayIndex, NewValue)
```

Voir aussi

Concepts

Séquence d'appel de procédure pour les utilisateurs de Visual Basic 6.0
Paramètres et arguments d'une procédure

La méthode '<nomméthode>' est déjà déclarée dans l'interface '<nominterface>'

Method '<methodname>' is already declared in interface '<interfacename>'

Vous avez déclaré une méthode d'interface une seconde fois en la dupliquant.

ID d'erreur : BC30377

Pour corriger cette erreur

- Supprimez la déclaration dupliquée.

Voir aussi

Concepts

Exemples d'implémentation d'interface dans Visual Basic

La méthode 'MustOverride' '<nomméthode>' ne peut pas être appelée par 'MyClass'

'MustOverride' method '<methodname>' cannot be called with 'MyClass'

MyClass équivaut à **Me**, mais tous les appels de méthodes sur celle-ci sont traités comme si la méthode appelée était **NotOverridable**.

ID d'erreur : BC30614

Pour corriger cette erreur

- Supprimez le modificateur **MustOverride** ou déclarez la méthode dans une classe de base, puis héritez et substituez cette classe.

Voir aussi

Référence

MustOverride
MustInherit
NotOverridable

'MustOverride' ne peut pas être spécifié pour '<NomProcédure>', car il est dans un type partiel déclaré 'NotInheritable' dans une autre définition partielle

'MustOverride' cannot be specified on '<procedurename>' because it is in a partial type that is declared 'NotInheritable' in another partial definition

Une procédure ou une propriété est déclarée comme **MustOverride** dans une classe définie dans plusieurs déclarations partielles, mais l'une des définitions partielles spécifie **NotInheritable** pour la classe.

Lorsque vous divisez la définition d'une classe en plusieurs déclarations partielles, le compilateur traite la classe comme l'union de toutes ses déclarations partielles. Cela s'applique non seulement aux membres, mais également à l'implémentation, l'héritage et le niveau d'accès.

Pour substituer une procédure ou une propriété, une classe doit en hériter d'une classe de base. Par conséquent, vous devez spécifier **MustInherit** pour la classe afin de spécifier **MustOverride** pour une procédure ou une propriété dans une classe de base. Ces mots clés étant mutuellement contradictoires, vous ne pouvez pas spécifier **MustInherit** et **NotInheritable** pour la même classe.

ID d'erreur : BC30927

Pour corriger cette erreur

- Si la propriété ou la procédure doit être substituée, supprimez le mot clé **NotInheritable** de la déclaration partielle dans laquelle il apparaît.
- Si la classe doit avoir la valeur **NotInheritable**, supprimez le mot clé **MustOverride** de la procédure ou la propriété. Vous ne pouvez pas le substituer car vous ne pouvez pas hériter de la classe.

Voir aussi

Référence

Partial (Visual Basic)
MustOverride
MustInherit
NotInheritable

Concepts

Éléments fondamentaux de l'héritage

'MustInherit' ne peut pas être spécifié pour le type partiel '<NomTypePartiel>', car il ne peut pas être combiné avec 'NotInheritable' spécifié pour l'un de ses autres types partiels

'MustInherit' cannot be specified for partial type '<partialtypename>' because it cannot be combined with 'NotInheritable' specified for one of its other partial types

Une classe est définie dans plusieurs déclarations partielles, dont une spécifie **MustInherit** et une autre **NotInheritable**.

Lorsque vous divisez la définition d'une classe en plusieurs déclarations partielles, le compilateur traite la classe comme l'union de toutes ses déclarations partielles. Cela s'applique non seulement aux membres, mais également à l'implémentation, l'héritage et le niveau d'accès.

Une classe ne peut pas être à la fois *abstract* et *sealed*, c'est-à-dire qu'elle ne peut pas exiger et interdire l'héritage. Par conséquent, vous ne pouvez pas spécifier **MustInherit** et **NotInheritable** pour la même classe.

ID d'erreur : BC30926

Pour corriger cette erreur

- Choisissez si la classe doit exiger l'héritage, interdire l'héritage, ou ni l'un ni l'autre, et supprimez les mots clés qui sont inappropriés par rapport à votre choix.

Voir aussi

Référence

Partial (Visual Basic)
MustInherit
NotInheritable

Concepts

Éléments fondamentaux de l'héritage

Les modules ne peuvent pas être génériques

Modules cannot be generic

Une instruction **Module** utilise le mot clé **Of** pour introduire des paramètres de type générique.

Vous pouvez définir et utiliser des classes, des structures, des interfaces, des procédures et des délégués génériques. Vous ne pouvez pas définir de modules génériques.

ID d'erreur : BC32073

Pour corriger cette erreur

1. Supprimez le mot clé **Of** de l'instruction **Module**.
2. Pour utiliser les fonctionnalités d'un module générique, l'approximation la plus proche est une classe générique. Utilisez Class, instruction (Visual Basic) au lieu d'une instruction **Module**.

Voir aussi

Tâches

Comment : définir une classe qui fournisse des fonctionnalités identiques pour différents types de données

Référence

Module, instruction

Of

Concepts

Types génériques en Visual Basic

Les modules ne peuvent pas être déclarés '<nommodificateur>'

Modules cannot be declared '<specifier>'

Un module a été déclaré avec un modificateur d'accès non valide.

ID d'erreur : BC31052

Pour corriger cette erreur

- Remplacez le modificateur d'accès par un modificateur d'accès valide, tel que **Public** ou **Friend**.

Voir aussi

Référence

Module, instruction

Les instructions 'Module' ne peuvent intervenir qu'au niveau du fichier ou de l'espace de noms

'Module' statements can occur only at file or namespace level

Les instructions **Module** doivent apparaître au début de votre fichier source, immédiatement après les instructions **Option** et **Imports**, les attributs globaux et les déclarations d'espaces de noms, mais avant toutes les autres déclarations.

ID d'erreur : BC30617

Pour corriger cette erreur

- Déplacez l'instruction **Module** au début de votre déclaration d'espace de noms ou de votre fichier source.

Voir aussi

Référence

Module, instruction

L'instruction 'Module' doit se terminer par un 'End Module' correspondant

'Module' statement must end with a matching 'End Module'

Les blocs module doivent se terminer par une instruction **End Module**.

ID d'erreur : BC30625

Pour corriger cette erreur

- Ajoutez une instruction **End Module** à la fin du bloc **Module**.

Voir aussi

Référence

End (Visual Basic)

Attribut de module '<nomattribut>' non valide : <erreur>

Module attribute '<attributename>' is not valid: <error>

Le Common Language Runtime n'accepte pas l'attribut sur un module.

ID d'erreur : BC30130

Pour corriger cette erreur

1. Supprimez l'attribut de la déclaration ou remplacez-le par un attribut valide pour le contexte.
2. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.

Voir aussi

Tâches

Comment : définir vos propres attributs

Le module '<nommodule>' ne peut pas être utilisé en tant que type

Module '<modulename>' cannot be used as a type

Vous avez tenté d'utiliser un module comme un type. Un module ne peut pas être utilisé, sauf en tant que type **Get**.

ID d'erreur : BC30371

Pour corriger cette erreur

- Modifiez le module par un élément pouvant être utilisé comme un type valide.

Voir aussi

Référence

Module, instruction

'Microsoft.VisualBasic.ComClassAttribute' dans la classe '<nomclasse>' déclare implicitement <type> '<nommembre>', qui est en conflit avec un membre portant le même nom dans <type> '<nomtype>'

'Microsoft.VisualBasic.ComClassAttribute' on class '<classname>' implicitly declares <type> '<membername>', which conflicts with a member of the same name in <type> '<typename>'

Une classe qui utilise un bloc d'attributs **COMClassAttribute** définit implicitement une interface portant le même nom qu'un membre de la classe de base. Dans ce cas, le nom de l'interface doit occulter le membre de la classe de base.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42101

Pour corriger cette erreur

1. Si vous souhaitez masquer le membre de la classe de base, définissez `InterfaceShadows:=True` dans le bloc d'attributs **ComClassAttribute**.
2. Dans le cas contraire, changez le nom de la classe.

Voir aussi

Référence

ComClassAttribute, classe
ComClassAttribute.InterfaceShadows, propriété

Concepts

Attributs utilisés dans Visual Basic
Application des attributs

'Microsoft.VisualBasic.ComClassAttribute' est spécifié pour la classe '<nomclasse>' mais il ne possède pas de membres publics pouvant être exposés à COM ; par conséquent, aucune interface COM n'est générée

'Microsoft.VisualBasic.ComClassAttribute' is specified for class '<classname>' but it has no public members that can be exposed to COM; therefore no COM interfaces are generated

Une classe qui utilise un bloc d'attributs **COMClassAttribute** ne définit pas de propriété ou de méthode **Public**. Si une classe doit être exposée en tant qu'objet COM, ses propriétés et méthodes doivent être déclarées avec un accès **Public**.

Le message est un avertissement par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40011

Pour corriger cette erreur

- Ajoutez le mot clé **Public** à une ou plusieurs propriétés ou méthodes de la classe, ou supprimez le bloc d'attributs **COMClassAttribute**.

Voir aussi

Référence

Public (Visual Basic)

ComClassAttribute, classe

Concepts

Attributs utilisés dans Visual Basic

Application des attributs

'Microsoft.VisualBasic.ComClassAttribute' ne peut pas être appliqué à une classe qui est générique ou contenue dans un type générique

'Microsoft.VisualBasic.ComClassAttribute' cannot be applied to a class that is generic or nested inside a generic type

Une classe est déclarée avec ComClassAttribute, mais elle est générique ou contenue dans une classe ou une structure générique.

Pour être prêt pour COM Interop, une classe .NET Framework doit remplir les conditions suivantes :

- elle doit être **Public**, tous ses conteneurs doivent être **Public** et il doit exposer au moins un membre **Public** ;
- elle ne doit pas être *abstraite*, c'est-à-dire qu'elle ne doit pas être déclarée avec **MustInherit** ;
- elle ne doit pas être générique ou déclarée dans un type de conteneur générique.

ID d'erreur : BC31527

Pour corriger cette erreur

- Modifiez la déclaration de la classe afin qu'elle ne soit pas générique, et vérifiez que son élément conteneur n'est pas générique.

  - ou -

- Si la classe ou son élément conteneur doit être générique, supprimez **ComClassAttribute** de la déclaration de classe. Vous ne pouvez pas l'exposer à COM.

Voir aussi

Référence
ComClassAttribute
Concepts
Types génériques en Visual Basic
Autres ressources
COM Interop

'Microsoft.VisualBasic.ComClassAttribute' ne peut pas être appliqué à une classe qui est déclarée 'MustInherit'

'Microsoft.VisualBasic.ComClassAttribute' cannot be applied to a class that is declared 'MustInherit'

Une classe est déclarée avec l'ComClassAttribute, mais sa déclaration spécifie **MustInherit**.

Pour être prêt pour COM Interop, une classe .NET Framework doit remplir les conditions suivantes :

- elle doit être **Public**, tous ses conteneurs doivent être **Public** et il doit exposer au moins un membre **Public** ;
- elle ne doit pas être *abstraite*, c'est-à-dire qu'elle ne doit pas être déclarée avec **MustInherit** ;
- elle ne doit pas être générique ou déclarée dans un type de conteneur générique.

ID d'erreur : BC32508

Pour corriger cette erreur

- Supprimez le mot clé **MustInherit** de la déclaration de classe.

  - ou -

- Si la classe ou son élément conteneur doit être générique, supprimez l'**ComClassAttribute** de la déclaration de classe. Vous ne pouvez pas l'exposer à COM.

Voir aussi

Référence

MustInherit

ComClassAttribute

Autres ressources

COM Interop

Impossible d'appliquer 'Microsoft.VisualBasic.ComClassAttribute' à '<nomclasse1>', car son conteneur '<nomclasse2>' n'est pas déclaré 'Public'

'Microsoft.VisualBasic.ComClassAttribute' cannot be applied to '<classname1>' because its container '<classname2>' is not declared 'Public'

Une classe qui utilise un bloc d'attributs **COMClassAttribute** est déclarée à l'intérieur d'une classe qui n'est pas **Public**. Si une classe doit être exposée en tant qu'objet COM, sa hiérarchie de relation contenant-contenu doit être déclarée avec un accès **Public**.

ID d'erreur : BC32504

Pour corriger cette erreur

- Déclarez toutes les classes conteneur **Public** ou supprimez le bloc d'attributs **COMClassAttribute**.

Voir aussi

Référence

ComClassAttribute, classe

Public (Visual Basic)

Concepts

Attributs utilisés dans Visual Basic

Application des attributs

Impossible d'appliquer 'Microsoft.VisualBasic.ComClassAttribute' à '<NomClasse>', car il n'est pas déclaré 'Public'

'Microsoft.VisualBasic.ComClassAttribute' cannot be applied to '<classname>' because it is not declared 'Public'

Une classe est déclarée avec ComClassAttribute, mais sa déclaration ne spécifie pas **Public**.

Pour prendre en charge COM Interop, une classe du .NET Framework doit répondre aux exigences suivantes :

- Elle doit être **Public**, tous ses conteneurs doivent être **Public**, et elle doit exposer au moins un membre **Public**.
- Elle ne doit pas être *abstraite*, c'est-à-dire qu'elle ne doit pas être déclarée avec **MustInherit**.
- Elle ne doit pas être générique ou être déclarée dans un type conteneur générique.

ID d'erreur : BC32509

Pour corriger cette erreur

- Ajoutez le mot clé **Public** à la déclaration de la classe.

  - ou -

- Si la classe ou son élément conteneur ne peut pas être **Public**, supprimez **ComClassAttribute** de la déclaration de la classe. Vous ne pouvez pas l'exposer à COM.

Voir aussi

Référence

Public (Visual Basic)
ComClassAttribute

Autres ressources

COM Interop

Impossible d'appliquer à la fois 'Microsoft.VisualBasic.ComClassAttribute' et '<attribut>' à la même classe

'Microsoft.VisualBasic.ComClassAttribute' and '<attribute>' cannot both be applied to the same class

Un bloc d'attributs **COMClassAttribute** est utilisé avec un attribut qui ne s'applique pas aux objets COM. L'une des causes possibles est la confusion entre attributs .NET Framework et attributs COM.

ID d'erreur : BC32501

Pour corriger cette erreur

- Supprimez le bloc d'attributs **COMClassAttribute** ou l'attribut qui ne s'applique pas à COM.

Voir aussi

Référence

ComClassAttribute, classe

Concepts

Attributs utilisés dans Visual Basic

Application des attributs

Les méthodes ou événements qui implémentent des membres d'interface ne peuvent pas être déclarés 'Shared'

Methods or events that implement interface members cannot be declared 'Shared'

Vous avez tenté de déclarer **Shared** une méthode ou un événement qui implémente un membre d'interface. Les méthodes et les événements implémentés dans une classe ne peuvent pas être désignés comme étant **Shared** ou **Private**, sauf dans une classe non héritable.

ID d'erreur : BC30505

Pour corriger cette erreur

- Supprimez le mot clé **Shared**.

Voir aussi

Référence

Shared (Visual Basic)

Les méthodes d'un module ne peuvent pas implémenter les membres d'interface

Methods in a Module cannot implement interface members

Les membres d'interface doivent être implémentés par des classes ou des structures.

ID d'erreur : BC31083

Pour corriger cette erreur

- Implémentez des membres d'interface à partir de classes ou de structures.

Voir aussi

Référence

Class, instruction (Visual Basic)
Structure, instruction
Implements, instruction
Interface, instruction (Visual Basic)

Les méthodes d'un module ne peuvent pas être déclarées '<spécificateur>'

Methods in a Module cannot be declared '<specifier>'

Vous avez utilisé un spécificateur non valide pour une méthode dans une instruction **Module**. Les modules ne peuvent jamais être instanciés, ne prennent pas en charge l'héritage et ne peuvent pas implémenter des interfaces.

ID d'erreur : BC30433

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Module, instruction

Les méthodes déclarées 'Overrides' ne peuvent pas être déclarées 'Overridable', car elles sont implicitement substituables (Overridable)

Methods declared 'Overrides' cannot be declared 'Overridable' because they are implicitly overridable

À la différence de la plupart des méthodes, les méthodes marquées avec le modificateur **Overrides** sont substituables par défaut.

ID d'erreur : BC30730

Pour corriger cette erreur

- Ôtez le modificateur **Overridable** des méthodes marquées avec le modificateur **Overrides**.

Voir aussi

Référence

Overrides
Overridable

Les méthodes déclarées dans des structures ne peuvent pas comporter de clauses 'Handles'

Methods declared in structures cannot have 'Handles' clauses

Les méthodes Structure ne peuvent pas utiliser le mot clé **Handles** pour gérer des événements.

ID d'erreur : BC30728

Pour corriger cette erreur

- Modifiez votre structure pour la transformer en classe.

  Vous pouvez utiliser **AddHandler** pour associer un événement à un gestionnaire d'événements dans une structure, à condition que la structure implémente un gestionnaire d'événements défini dans une interface.

Voir aussi

Concepts

Structures et classes
Classes : modèles d'objets
AddHandler et RemoveHandler

Autres ressources

Événements en Visual Basic

Les méthodes ne peuvent pas être déclarées 'Static'

Methods cannot be declared 'Static'

Vous ne pouvez pas appliquer le modificateur **Static** à des procédures entières.

ID d'erreur : BC30810

Pour corriger cette erreur

- Utilisez le modificateur **Static** pour déclarer des éléments à l'intérieur d'une procédure en tant que static.

Voir aussi

Référence

Static (Visual Basic)

Le nom '<nom>' n'est pas déclaré

Name '<name>' is not declared

Une instruction fait référence à un élément de programmation, mais le compilateur ne peut pas trouver d'élément avec ce nom exact.

ID d'erreur : BC30451

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom dans l'instruction de référence. Visual Basic ne respecte pas la casse, mais considère toute autre variation orthographique comme un nom complètement différent. Notez que le trait de soulignement (_) fait partie du nom et, par conséquent, de l'orthographe.
2. Vérifiez que vous avez l'opérateur d'accès au membre (**.**) entre un objet et son membre. Par exemple, si vous avez un contrôle TextBox nommé `TextBox1`, vous devez, pour accéder à sa propriété Text, taper `TextBox1.Text`. Si, au lieu de cela, vous tapez `TextBox1Text`, vous avez créé un nom différent.
3. Si l'orthographe est correcte et que la syntaxe de l'accès au membre objet est correcte, vérifiez que l'élément a été déclaré. Pour plus d'informations, consultez Éléments déclarés en Visual Basic.
4. Si l'élément de programmation a été déclaré, vérifiez qu'il est dans la portée. Si l'instruction de référence se trouve à l'extérieur de la région qui déclare l'élément de programmation, vous devrez peut-être qualifier le nom d'élément. Pour plus d'informations, consultez Portée dans Visual Basic.

Voir aussi

Référence

Liste des déclarations et des constantes

Concepts

Conventions d'affectation de noms Visual Basic

Noms d'éléments déclarés

Autres ressources

Références aux éléments déclarés

Le nom <NomMembre> n'est pas conforme CLS

Name <membername> is not CLS-compliant

Un assembly est marqué comme `<CLSCompliant(True)>` mais expose un membre portant un nom qui commence par un trait de soulignement (_).

Un élément de programmation peut contenir un ou plusieurs traits de soulignement, mais pour être conforme à la Spécification CLS (Common Language Specification) (CLS), il ne doit pas commencer par un trait de soulignement. Consultez Noms d'éléments déclarés.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40031

Pour corriger cette erreur

- Si vous avez un contrôle sur le code source, modifiez le nom de membre afin qu'il ne commence pas par un trait de soulignement.
- S'il est nécessaire que le nom de membre reste inchangé, supprimez l'**CLSCompliantAttribute** de sa définition ou marquez-le comme `<CLSCompliant(False)>`. Vous pouvez toujours marquer l'assembly comme `<CLSCompliant(True)>`.

Voir aussi

Concepts

Noms d'éléments déclarés
Conventions d'affectation de noms Visual Basic
Écriture d'un code conforme CLS

Le nom <NomEspaceDeNoms> de l'espace de noms racine <EspaceDeNomsComplet> n'est pas conforme CLS

Name <namespacename> in the root namespace <fullnamespacename> is not CLS-compliant

Un assembly est marqué comme `<CLSCompliant(True)>`, mais un élément du nom de l'espace de noms racine commence par un trait de soulignement (_).

Un élément de programmation peut contenir un ou plusieurs traits de soulignement, mais pour être conforme à la Spécification CLS (Common Language Specification) (CLS), il ne doit pas commencer par un trait de soulignement. Consultez Noms d'éléments déclarés.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40039

Pour corriger cette erreur

- S'il est nécessaire d'avoir la conformité CLS, modifiez le nom de l'espace de noms racine pour qu'aucun de ses éléments ne commence par un trait de soulignement.
- Si le nom de l'espace de noms doit rester inchangé, supprimez l'**CLSCompliantAttribute** de l'assembly ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Tâches

Comment : modifier l'espace de noms pour une application

Référence

Namespace, instruction

/rootnamespace

Concepts

Espaces de noms dans Visual Basic

Noms d'éléments déclarés

Conventions d'affectation de noms Visual Basic

Écriture d'un code conforme CLS

'MyClass' doit être suivie de '.' et d'un identificateur

'MyClass' must be followed by '.' and an identifier

MyClass n'est pas une véritable variable objet et ne peut pas s'afficher seule. Vous ne l'utilisez que pour accéder à un membre de l'instance en cours comme si **NotOverridable** était utilisé dans la classe de base.

ID d'erreur : BC32028

Pour corriger cette erreur

1. Si vous souhaitez accéder à un membre de classe, spécifiez l'opérateur d'accès au membre (**.**) ainsi qu'un membre de l'instance en cours après **MyClass**.
2. Dans le cas contraire, utilisez le mot clé **Me**.

Voir aussi

Référence

MyClass
Me
NotOverridable

Concepts

Éléments fondamentaux de l'héritage

'MyClass' ne peut pas être utilisé à l'extérieur d'une classe

'MyClass' cannot be used outside of a class

Vous avez utilisé le mot clé **MyClass** à l'extérieur d'une structure de classe.

ID d'erreur : BC30470

Pour corriger cette erreur

- Placez la déclaration **MyClass** dans votre structure de classe.

Voir aussi

Référence

MyClass

Concepts

Substitution de propriétés et de méthodes

'MyBase' doit être suivie de '.' et d'un identificateur

'MyBase' must be followed by '.' and an identifier

MyBase n'est pas une véritable variable objet et ne peut pas s'afficher seule. Vous ne pouvez l'utiliser que pour accéder à un membre de la classe de base de l'instance en cours.

ID d'erreur : BC32027

Pour corriger cette erreur

- Si vous souhaitez un accès au membre, spécifiez l'opérateur d'accès au membre (.) ainsi qu'un membre de la classe de base après **MyBase**.
- Dans le cas contraire, déclarez et initialisez une instance de la classe de base, ou supprimez la référence à **MyBase**.

Voir aussi

Référence

MyBase

Concepts

Éléments fondamentaux de l'héritage

'MyBase' ne peut pas être utilisé avec la méthode '<nomméthode>', car il est déclaré 'MustOverride'

'MyBase' cannot be used with method '<methodname>' because it is declared 'MustOverride'

Vous avez tenté d'utiliser **MyBase** avec une méthode déclarée **MustOverride**.

ID d'erreur : BC30399

Pour corriger cette erreur

- Supprimez la déclaration **MustOverride**.

Voir aussi

Référence

MyBase
MustOverride

Argument nommé attendu

Named argument expected

Une liste d'arguments dans un appel de procédure passe un argument par nom, puis un argument par position. Tous les arguments positionnels doivent précéder ceux passés par nom dans un appel de procédure.

ID d'erreur : BC30241

Pour corriger cette erreur

- Rédigez de nouveau la liste des arguments afin de placer les arguments passés par position avant ceux passés par nom.

Voir aussi

Concepts

Passage des arguments par position et par nom

L'argument nommé ne peut pas correspondre à un paramètre ParamArray

Named argument cannot match a ParamArray parameter

Vous avez fourni un argument nommé (spécifié par le nom déclaré de l'argument suivi de deux-points et d'un signe égal (:=), suivi par la valeur de l'argument) ; vous ne pouvez toutefois pas passer un tableau de paramètres par un nom. Lorsque vous appelez la procédure, vous devez fournir un nombre indéfini d'arguments séparés par des virgules pour le tableau de paramètres, or, le compilateur ne peut pas associer plusieurs arguments à un seul nom.

ID d'erreur : BC30587

Pour corriger cette erreur

- Passez l'argument par position plutôt que par nom.

Voir aussi

Référence

ParamArray

Concepts

Passage des arguments par position et par nom

Les arguments nommés ne sont pas valides en tant qu'indices de tableau

Named arguments are not valid as array subscripts

Un index de tableau est spécifié à l'aide de la syntaxe de passage d'un argument de procédure par nom, par exemple `Array(Index := 10)`. Cette syntaxe n'est pas valide pour les indices de tableau.

ID d'erreur : BC30075

Pour corriger cette erreur

- Utilisez une variable ordinaire ou une expression constante en tant qu'index de tableau, par exemple `Array(10)`.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Passage des arguments par position et par nom

L'espace de noms ou le type '<nom>' a déjà été importé

Namespace or type '<name>' has already been imported

Une tentative d'importation d'un espace de noms a été effectuée plusieurs fois.

ID d'erreur : BC31051

Pour corriger cette erreur

- Supprimez les instructions **Imports** redondantes.

Voir aussi

Référence

Imports, instruction

L'espace de noms ou le type spécifié dans les Imports '<NomÉlémentQualifié>' ne contient aucun membre public ou est introuvable

Message d'erreur
L'espace de noms ou le type spécifié dans les Imports '<NomÉlémentQualifié>' ne contient aucun membre public ou est introuvable Vérifiez que l'espace de noms ou le type est défini et qu'il contient au moins un membre public. Vérifiez que le nom d'alias ne contient pas d'autres alias.
Namespace or type specified in the Imports '<qualifiedelementname>' doesn't contain any public member or cannot be found. Make sure the namespace or the type is defined and contains at least one public member. Make sure the alias name doesn't contain other aliases.

Une instruction **Imports** spécifie un élément conteneur qui soit est introuvable, soit ne définit pas de membres **Public**.

Un *élément conteneur* peut être un espace de noms, une classe, une structure, un module, une interface ou une énumération. L'élément conteneur contient des membres, comme des variables, des procédures ou d'autres éléments conteneur.

L'objectif de l'importation est de permettre à votre code d'accéder à des membres espace de noms ou type sans devoir les qualifier. Votre projet peut également avoir besoin d'ajouter une référence à l'espace de noms ou au type. Pour plus d'informations, consultez « Importation d'éléments conteneurs » dans
Résolution d'une référence lorsque plusieurs variables ont le même nom.

Si le compilateur ne trouve pas l'élément conteneur spécifié, il ne peut pas résoudre les références qui l'utilisent. S'il trouve l'élément mais que celui-ci n'expose pas de membres **Public**, aucune référence ne peut aboutir. Dans l'un et l'autre cas, l'importation de l'élément n'a aucun sens.

Pensez que si vous importez un élément contenant et que vous lui assignez un alias d'importation, vous ne pouvez pas utiliser cet alias pour importer un autre élément. Le code suivant génère une erreur de compilation.

```
Imports winfrm = System.Windows.Forms

' The following statement is INVALID because it reuses an import alias.

Imports behav = winfrm .Design.Behavior
```

ID d'erreur : BC40056

Pour corriger cette erreur

1. Vérifiez que l'élément conteneur est accessible à partir de votre projet.
2. Vérifiez que la spécification de l'élément conteneur ne contient aucun alias d'importation provenant d'une autre importation.
3. Vérifiez que l'élément conteneur expose au moins un membre **Public**.

Voir aussi

Référence
Imports, instruction
Namespace, instruction
Public (Visual Basic)
Concepts
Espaces de noms dans Visual Basic
Résolution d'une référence lorsque plusieurs variables ont le même nom

L'espace de noms ou le type spécifié dans les Imports '<NomÉlémentQualifié>' au niveau du projet ne contient aucun membre public ou est introuvable

Message d'erreur
L'espace de noms ou le type spécifié dans les Imports '<NomÉlémentQualifié>' au niveau du projet ne contient aucun membre public ou est introuvable Vérifiez que l'espace de noms ou le type est défini et qu'il contient au moins un membre public. Vérifiez que le nom d'alias ne contient pas d'autres alias.
Namespace or type specified in the project-level Imports '<qualifiedelementname>' doesn't contain any public member or cannot be found. Make sure the namespace or the type is defined and contains at least one public member. Make sure the alias name doesn't contain other aliases.

Une propriété Imports d'un projet spécifie un élément conteneur qui soit ne peut pas être recherché soit ne définit pas de membres **Public**.

Un *élément conteneur* peut être un espace de noms, une classe, une structure, un module, une interface ou une énumération. L'élément conteneur contient des membres, comme des variables, des procédures ou d'autres éléments conteneur.

L'objectif de l'importation est de permettre à votre code d'accéder à des membres espace de noms ou type sans devoir les qualifier. Votre projet peut également avoir besoin d'ajouter une référence à l'espace de noms ou au type. Pour plus d'informations, consultez « Importation d'éléments conteneurs » dans
Résolution d'une référence lorsque plusieurs variables ont le même nom.

Si le compilateur ne trouve pas l'élément conteneur spécifié, il ne peut pas résoudre les références qui l'utilisent. S'il trouve l'élément mais que celui-ci n'expose pas de membres **Public**, aucune référence ne peut aboutir. Dans l'un et l'autre cas, l'importation de l'élément n'a aucun sens.

Vous pouvez utiliser le **Concepteur de projets** pour spécifier des éléments à importer. Utilisez la section **Espaces de noms importés** de la page **Références**. Vous pouvez ouvrir le **Concepteur de projets** en double-cliquant sur l'icône **Mon projet** dans l'**Explorateur de solutions**.

ID d'erreur : BC40057

Pour corriger cette erreur

1. Ouvrez le **Concepteur de projets** et basculez vers la page **Référence**.
2. Dans la section **Espaces de noms importés**, vérifiez que l'élément conteneur est accessible à partir de votre projet.
3. Vérifiez que l'élément conteneur expose au moins un membre **Public**.

Voir aussi

Tâches
Comment : modifier des propriétés de projet et des paramètres de configuration
Référence
Public (Visual Basic)
Concepts
Propriétés du projet
Espaces de noms dans Visual Basic
Résolution d'une référence lorsque plusieurs variables ont le même nom

L'instruction 'Namespace' doit se terminer par un 'End Namespace' correspondant

'Namespace' statement must end with a matching 'End Namespace'

Les blocs **Namespace** doivent se terminer par une instruction **End Namespace**.

ID d'erreur : BC30626

Pour corriger cette erreur

- Ajoutez une instruction **End Namespace** à la fin du bloc **Namespace**.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

Les instructions 'Namespace' ne peuvent intervenir qu'au niveau du fichier ou de l'espace de noms

'Namespace' statements can occur only at file or namespace level

Les instructions **Namespace** doivent apparaître après les instructions **Option**, **Imports** et les attributs globaux, mais avant toutes les autres déclarations de votre fichier source.

ID d'erreur : BC30618

Pour corriger cette erreur

- Déplacez l'instruction **Namespace** au début de votre déclaration d'espace de noms ou de votre fichier source.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

Aucune méthode accessible '<NomProcédure>' ne correspond à la signature du délégué '<NomDélégué>' : <SousListeErreurs>

No accessible method '<procedurename>' matches the signature of delegate '<delegatename>':<suberrorlist>

Une instruction d'assignation assigne l'adresse d'une procédure à une variable de délégué, mais le compilateur ne peut pas trouver une version de la procédure avec une signature correspondante.

Lorsque le code utilise l'adresse d'une procédure, le compilateur essaie de rechercher une version de cette procédure avec une liste de paramètres correspondant à celle du délégué. Si la procédure est définie dans plusieurs versions surchargées, le compilateur essaie de rechercher une version unique avec une signature correspondante. Pour plus d'informations, consultez Résolution de surcharge.

Si le compilateur ne peut pas trouver une version de la procédure avec une signature correspondante, il génère cette erreur. Par exemple, cela peut arriver si la procédure ou le délégué est générique et qu'un argument de type lui est transmis et lui donne une signature qui ne correspond pas à l'autre signature.

ID d'erreur : BC30950

Pour corriger cette erreur

1. Redéfinissez la procédure ou le délégué afin que leurs listes de paramètres correspondent.

   - ou -

   Définissez un nouveau délégué dont la liste de paramètres correspond à celle de la procédure ou définissez une nouvelle procédure dont la liste de paramètres correspond à celle du délégué.

2. Si la procédure ou le délégué est générique, passez-lui un argument de type afin que sa signature corresponde à l'autre signature.

Voir aussi

Référence

AddressOf, opérateur
Delegate, instruction

Concepts

Délégués et opérateur AddressOf
Résolution de surcharge
Types génériques en Visual Basic

Aucune méthode 'Main' accessible avec une signature appropriée n'a été trouvée dans '<nom>'

No accessible 'Main' method with an appropriate signature was found in '<name>'

Les applications en ligne de commande doivent avoir un **Sub Main** défini. **Main** doit être déclaré en tant que **Public Shared** s'il est défini dans une classe, ou en tant que **Public** s'il est défini dans un module.

Pour plus d'informations sur **Main**, consultez Version Visual Basic de Hello World!.

ID d'erreur : BC30737

Pour corriger cette erreur

- Définissez une procédure **Public Sub Main** pour votre projet. Déclarez-la en tant que **Shared** si et seulement si vous la définissez au sein d'une classe.

Voir aussi

Référence

Structure d'un programme Visual Basic

Concepts

Version Visual Basic de Hello World!

Procédures dans Visual Basic

Aucun '<NomProcédure>' accessible n'est plus spécifique : <ListeSignatures>

No accessible '<procedurename>' is most specific: <signaturelist>

Une instruction d'assignation assigne l'adresse d'une procédure surchargée à une variable de délégué, mais le compilateur ne peut pas résoudre plusieurs versions surchargées.

Lorsque le code utilise l'adresse d'une procédure qui est définie dans plusieurs versions surchargées, le compilateur doit choisir la surcharge à utiliser. Il essaie de rechercher une version unique dont la liste des paramètres correspond à celle du délégué. Pour plus d'informations, consultez Résolution de surcharge.

Si le compilateur trouve plusieurs versions de la procédure avec une signature correspondante, il génère cette erreur. Cela peut se produire, par exemple, si l'une des surcharges est générique et si l'argument de type qui lui est passé fournit une signature identique à celle d'une autre surcharge.

ID d'erreur : BC30794

Pour corriger cette erreur

- Si le conflit est dû à une surcharge générique dont la signature est identique à celle d'une autre surcharge, modifiez l'argument de type passé à cette surcharge générique.

Voir aussi

Référence

AddressOf, opérateur
Delegate, instruction

Concepts

Délégués et opérateur AddressOf
Résolution de surcharge
Types génériques en Visual Basic

Aucun '<NomProcédure>' accessible non générique trouvé

No accessible non-generic '<procedurename>' found

Une instruction appelle une procédure qui contient plusieurs versions surchargées, mais toutes les versions surchargées sont génériques et l'appel ne fournit pas d'arguments de type.

S'il existe une seule version générique et si vous l'appelez sans arguments de type, le compilateur peut tenter une *inférence du type*. Pour plus d'informations, consultez « Inférence du type » dans Procédures génériques dans Visual Basic. Toutefois, s'il existe plusieurs versions génériques, le compilateur ne peut pas choisir parmi elles sauf si vous fournissez des arguments de type. Si vous fournissez un argument de type, vous devez le faire pour chaque paramètre de type défini par l'une des versions surchargées.

ID d'erreur : BC32117

Pour corriger cette erreur

- Appelez la procédure avec une liste d'arguments de type.

Voir aussi

Référence

Overloads

Concepts

Surcharge de procédure

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Aucun '<NomProcédureGénérique>' accessible n'accepte ce nombre d'arguments de type

No accessible '<genericprocedurename>' accepts this number of type arguments

Une instruction appelle une procédure générique qui contient plusieurs versions surchargées, mais aucune des versions surchargées ne définit le même nombre de paramètres de type que le nombre d'arguments de type fournis dans l'appel.

S'il existe une seule version générique, vous l'appelez sans arguments de type, et le compilateur peut tenter une *inférence du type*. Pour plus d'informations, consultez « Inférence du type » dans Procédures génériques dans Visual Basic. Toutefois, s'il existe plusieurs versions génériques, le compilateur ne peut pas choisir parmi elles sauf si vous fournissez des arguments de type. Si vous fournissez un argument de type, vous devez le faire pour chaque paramètre de type défini par l'une des versions surchargées.

ID d'erreur : BC32118

Pour corriger cette erreur

- Choisissez la version surchargée que vous souhaitez appeler, puis fournissez le nombre approprié d'arguments de type.

Voir aussi

Référence

Overloads

Concepts

Surcharge de procédure
Types génériques en Visual Basic
Procédures génériques dans Visual Basic

L'instruction 'Next' nomme plus de variables qu'il n'existe d'instructions 'For' correspondantes

'Next' statement names more variables than there are matching 'For' statements

Les boucles imbriquées sont terminées par une seule instruction **Next** qui spécifie davantage de variables de boucle qu'il n'existe de boucles dans l'imbrication, comme dans l'exemple suivant :

```
For I = 1 To 10
   For J = 1 To 5
      ' ...
Next J, I, K   ' Next J, I is valid, but there is no loop on K.
```

ID d'erreur : BC32037

Pour corriger cette erreur

1. Assurez-vous que l'instruction **Next** spécifie toutes les variables de boucle imbriquées dans l'ordre inverse de leur initiation.
2. Supprimez les variables superflues de l'instruction **Next**.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

'Next' doit être précédé d'un 'For' correspondant

'Next' must be preceded by a matching 'For'

Une instruction **Next** apparaît sans instruction **For** ou **For Each** correspondante. Une instruction **Next** doit être précédée d'une instruction **For** ou **For Each** correspondante.

ID d'erreur : BC30092

Pour corriger cette erreur

1. Si cette boucle **For** fait partie d'un ensemble de boucles **For** imbriquées, vérifiez que chaque boucle est correctement terminée.
2. Vérifiez que les autres structures de contrôle à l'intérieur de la boucle **For** sont correctement terminées.
3. Vérifiez que cette boucle **For** est correctement mise en forme.

Voir aussi

Référence

For...Next, instruction (Visual Basic)
For Each...Next, instruction (Visual Basic)

'Next' attendu

'Next' expected

Une construction **On Error Resume** se produit sans **Next** de terminaison.

ID d'erreur : BC30003

Pour corriger cette erreur

- Ajoutez le mot clé **Next** à la fin de l'instruction **On Error**.

Voir aussi

Référence

On Error, instruction (Visual Basic)
Error, instruction
Resume, instruction

Concepts

Vue d'ensemble de la gestion non structurée des exceptions

La variable de contrôle Next ne correspond pas à la variable de contrôle de boucle For '<nomvariable>'

Next control variable does not match For loop control variable '<variablename>'

La variable de contrôle de l'instruction **Next** d'une boucle **For...Next** doit correspondre à la variable de l'instruction **For** correspondante.

ID d'erreur : BC30070

Pour corriger cette erreur

1. Vérifiez que l'orthographe de la variable dans l'instruction **Next** et celle de la variable dans l'instruction **For** correspondante sont identiques.
2. Vérifiez qu'aucune partie de la boucle englobante n'a été supprimée par inadvertance.
3. Si cette boucle fait partie d'un ensemble de boucles imbriquées, vérifiez que chaque boucle est correctement terminée.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

'New' n'est pas valide dans ce contexte

'New' is not valid in this context

Le mot clé **New** apparaît là où un nom de type est requis. Le contexte exige un type valeur qui ne peut pas être combiné avec **New**.

ID d'erreur : BC30200

Pour corriger cette erreur

- Supprimez le mot clé **New**.

Voir aussi

Référence

New (Visual Basic)
Enum, instruction (Visual Basic)

La contrainte 'New' ne peut pas être spécifiée plusieurs fois pour le même paramètre de type

'New' constraint cannot be specified multiple times for the same type parameter

Une liste de contraintes contient plusieurs fois la contrainte New (Visual Basic).

Une liste de contraintes sur un paramètre de type peut spécifier que l'argument de type passé à ce paramètre de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder. Un type ne peut pas contenir plusieurs constructeurs sans paramètre, et vous ne pouvez pas spécifier plusieurs fois cette exigence.

ID d'erreur : BC32081

Pour corriger cette erreur

- Supprimez les mots clés **New** redondants. La liste de contraintes ne doit en contenir qu'un seul.

Voir aussi

Concepts

Types génériques en Visual Basic

La contrainte 'New' et la contrainte 'Structure' ne peuvent pas être combinées

'New' constraint and 'Structure' constraint cannot be combined

Une liste de contraintes contient la contrainte New (Visual Basic) et la contrainte Structure (Visual Basic).

Une liste de contraintes sur un paramètre de type peut spécifier que l'argument de type passé à ce paramètre de type doit être un type valeur (avec la contrainte **Structure**) ou un type référence (avec la contrainte Class (Visual Basic)). Vous ne pouvez pas spécifier les deux contraintes sur le même paramètre de type, de même que vous ne pouvez pas spécifier l'une ou l'autre plusieurs fois.

La contrainte **New** spécifie que l'argument de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder. Toutefois, une structure ne peut pas contenir de constructeur sans paramètre non partagé. Par conséquent, les contraintes **New** et **Structure** sont en conflit.

ID d'erreur : BC32103

Pour corriger cette erreur

1. Choisissez si un type valeur ou un type référence doit être requis pour l'argument de type.
2. Si vous souhaitez que l'argument de type soit un type valeur, supprimez le mot clé **New** de la liste des contraintes.
3. Si vous souhaitez que l'argument de type soit un type référence, supprimez le mot clé **Structure** de la liste des contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

'New' ne peut pas être utilisé pour la classe '<nomclasse>', car elle contient un membre 'MustOverride' qui n'a pas été substitué

'New' cannot be used on class '<classname>' because it contains a 'MustOverride' member that has not been overridden

Vous avez tenté d'utiliser **New** dans une classe qui contient un membre **MustOverride** qui n'a pas été substitué.

ID d'erreur : BC30376

Pour corriger cette erreur

- Supprimez l'instruction **New**.

Voir aussi

Référence

MustOverride

Impossible d'utiliser 'New' pour une interface

'New' cannot be used on an interface

Dim, instruction (Visual Basic) utilise une clause New (Visual Basic) pour déclarer une variable comme étant d'un type interface.

Bien qu'une interface soit un type référence, vous ne pouvez pas créer d'instance de celle-ci. Vous pouvez uniquement utiliser **New** pour créer une instance d'une classe ou d'une structure.

ID d'erreur : BC30375

Pour corriger cette erreur

1. Si la variable doit être d'un type interface, supprimez le mot clé **New**.
2. Si la variable doit faire référence à une instance, déclarez-la comme étant d'une classe ou d'un type structure. Conservez le mot clé **New** pour créer une instance.

Voir aussi

Référence

Interface, instruction (Visual Basic)
Class, instruction (Visual Basic)
Structure, instruction

'New' ne peut pas être utilisé pour un paramètre de type qui n'a pas de contrainte 'New'

'New' cannot be used on a type parameter that does not have a 'New' constraint

Une instruction de déclaration utilise une clause New (Visual Basic) qui spécifie un paramètre de type comme type à créer, et le paramètre de type est déclaré sans contrainte **New**.

Une *contrainte* sur un paramètre de type impose des exigences sur tout argument de type passé à ce paramètre de type lors de la création du type générique. La contrainte **New** spécifie que l'argument de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder. Cela permet à une clause **New** dans une instruction de déclaration de créer une instance de ce type.

ID d'erreur : BC32046

Pour corriger cette erreur

- Si l'argument de type doit exposer un constructeur sans paramètre accessible, ajoutez la contrainte **New** à la déclaration du paramètre de type.
- Si l'argument de type ne doit pas exposer un constructeur sans paramètre accessible, supprimez la clause **New** de l'instruction de déclaration. Vous ne pouvez pas garantir qu'un argument de type passé à ce paramètre de type autorise la création d'une instance.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

'New' ne peut pas être utilisé dans une classe déclarée 'MustInherit'

'New' cannot be used on a class that is declared 'MustInherit'

Une classe **MustInherit** ne peut pas être instanciée directement. Par conséquent, l'opérateur **New** ne peut pas être utilisé dans une classe **MustInherit**. S'il est possible pour des variables et des valeurs d'avoir des types de compilation **MustInherit**, ces variables et valeurs équivaudront nécessairement à une valeur null ou contiendront des références à des instances de classes normales dérivées des types **MustInherit**.

ID d'erreur : BC30569

Pour corriger cette erreur

- Supprimez l'opérateur **New**.

Voir aussi

Référence

MustInherit
New (Visual Basic)

Aucune source d'entrée spécifiée

No input sources specified

Lors de la compilation, aucune source d'entrée n'a été spécifiée.

ID d'erreur : BC2008

Pour corriger cette erreur

- Assurez-vous que vous avez inclu un fichier de code source.

Voir aussi

Tâches

Comment : compiler et exécuter un projet dans Visual Basic

Le nombre d'indices est inférieur au nombre de dimensions du tableau indexé

Number of indices is less than the number of dimensions of the indexed array

Le nombre d'indices utilisé pour accéder à un élément de tableau doit être exactement identique au rang du tableau, c'est-à-dire au nombre de dimensions déclaré pour celui-ci.

ID d'erreur : BC30105

Pour corriger cette erreur

- Ajoutez des indices à la référence de tableau jusqu'à ce que le nombre total d'indices soit égal au rang du tableau.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Le nombre d'indices est supérieur au nombre de dimensions du tableau indexé

Number of indices exceeds the number of dimensions of the indexed array

Le nombre d'indices utilisé pour accéder à un élément de tableau doit être exactement identique au rang du tableau, c'est-à-dire au nombre de dimensions déclaré pour celui-ci.

ID d'erreur : BC30106

Pour corriger cette erreur

- Supprimez des indices de la référence de tableau jusqu'à ce que le nombre total d'indices soit égal au rang du tableau.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

'NotOverridable' ne peut pas être spécifié pour les méthodes qui ne se substituent pas à une autre méthode

'NotOverridable' cannot be specified on methods that do not override another method

Par défaut, les propriétés et les méthodes sont **NotOverridable**. Le modificateur **NotOverridable** ne peut être utilisé que sur les méthodes qui se substituent à une autre propriété ou une autre méthode.

ID d'erreur : BC31088

Pour corriger cette erreur

- Supprimez le modificateur **NotOverridable** des propriétés et des méthodes qui ne se substituent pas à un autre membre.

Voir aussi

Référence

Overrides
Overloads
NotOverridable
Overridable

Les classes 'NotInheritable' ne peuvent pas avoir de membres déclarés '<nomspécificateur>'

'NotInheritable' classes cannot have members declared '<specifiername>'

Vous ne pouvez pas utiliser les modificateurs override avec les classes **NotInheritable** car leurs membres ne peuvent pas être substitués.

ID d'erreur : BC30607

Pour corriger cette erreur

- Supprimez les modificateurs override, tels que **Overridable**, **NotOverridable** ou **MustOverride** de la définition de classe.

Voir aussi

Référence

Overridable
NotOverridable
MustOverride

Concepts

Substitution de propriétés et de méthodes

Impossible d'évaluer 'Nothing'

'Nothing' cannot be evaluated

Le mot clé **Nothing** a été utilisé dans la fenêtre de débogage.

ID d'erreur : BC30701

Pour corriger cette erreur

- N'utilisez pas **Nothing** lors du débogage.

Voir aussi

Référence

Nothing (Visual Basic)

Les membres non partagés dans une Structure ne peuvent pas être déclarés 'New'

Non-shared members in a Structure cannot be declared 'New'

Une variable non partagée dans une structure est déclarée avec une clause **New**.

Vous pouvez initialiser une variable de référence partagée dans une structure, et vous pouvez utiliser une variable de référence non partagée sans l'initialiser, comme le montrent les lignes de code suivantes.

```
Shared structVar1 As New System.ApplicationException

Dim structVar2 As System.ApplicationException
```

En revanche, vous ne pouvez pas initialiser une variable de référence non partagée dans une structure. La ligne de code suivante n'est pas valide.

```
Dim structVar3 As New System.ApplicationException ' INVALID IN A STRUCTURE
```

ID d'erreur : BC30795

Pour corriger cette erreur

- Supprimez le modificateur **Shared** ou le mot clé **New** de la déclaration de la variable de référence.

Voir aussi

Référence

Structure, instruction
Shared (Visual Basic)
New (Visual Basic)

L'attribut 'NonSerialized' n'aura pas d'effet sur ce membre, car sa classe conteneur n'est pas exposée comme 'Serializable'

'NonSerialized' attribute will not affect this member because its containing class is not exposed as 'Serializable'

Par défaut, les classes et leurs membres ne sont pas sérialisables. L'attribut NonSerializedAttribute n'est nécessaire que si un membre d'une classe sérialisable ne doit pas être sérialisé.

ID d'erreur : BC30772

Pour corriger cette erreur

- Ajoutez l'attribut SerializableAttribute à la classe.

  - ou -

- Supprimez l'attribut **NonSerializedAttribute** du membre.

Voir aussi

Référence

NonSerializedAttribute

SerializableAttribute

Autres ressources

Attributs en Visual Basic

Les noms de types non intrinsèques ne sont pas autorisés dans les expressions de compilation conditionnelles

Non-intrinsic type names are not allowed in conditional compilation expressions

L'expression de compilation conditionnelle fait référence à un type non intrinsèque, mais seuls les types intrinsèques sont autorisés.

Pour plus d'informations sur le type intrinsèque dans Visual Basic, consultez Liste des types de données.

ID d'erreur : BC31426

Pour corriger cette erreur

- Remplacez le type non intrinsèque par un type intrinsèque.

Voir aussi

Référence

Liste des types de données

Autres ressources

Compilation conditionnelle (Visual Basic)

Aucune des méthodes 'Main' accessibles avec les signatures appropriées ayant été trouvées dans '<NomType>' ne peut être la méthode de démarrage, car elles sont toutes génériques ou imbriquées dans des types génériques

None of the accessible 'Main' methods with the appropriate signatures found in '<typename>' can be the startup method since they are either generic or nested in generic types

Une classe, un module, ou une structure ne contient pas de procédure **Main** qui peut être considérée comme procédure de démarrage du projet.

Visual Basic requiert que la procédure de démarrage d'un projet ne soit pas dépendante des arguments de type. Par conséquent, elle doit être en mesure d'accéder au moins à une procédure **Main** qui n'est ni générique ni contenue dans un type générique.

ID d'erreur : BC30796

Pour corriger cette erreur

- Définissez au moins l'une des procédures **Main** afin qu'elle ne soit ni générique ni contenue dans un type générique.

  - ou -

- Sur la page **Propriétés** pour votre projet, spécifiez un autre formulaire ou module pour le **Formulaire de démarrage** ou l'**Objet de démarrage**.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Concepts

Types génériques en Visual Basic

Version Visual Basic de Hello World!

Le membre 'MustOverride' non conforme CLS n'est pas autorisé dans un <NomClasse> conforme CLS

Non-CLS-compliant 'MustOverride' member is not allowed in a CLS-compliant <classname>

Une classe est marquée comme `<CLSCompliant(True)>`, mais elle contient une propriété ou une procédure **MustOverride** qui est marquée comme `<CLSCompliant(False)>` ou qui n'est pas marquée.

Lorsqu'une classe est conforme au Spécification CLS (Common Language Specification) (CLS), une application qui l'utilise accède uniquement aux membres qui sont également marqués comme `<CLSCompliant(True)>` et ignore les membres qui ne le sont pas. Toutefois, l'application ne peut pas ignorer une propriété ou une procédure **MustOverride**, car elle doit y accéder pour la substituer.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40034

Pour corriger cette erreur

- Si la conformité CLS est requise et si vous avez le contrôle du code source de la classe, marquez le membre comme `<CLSCompliant(True)>`.
- Si la conformité CLS est requise et si vous n'avez pas le contrôle du code source de la classe ou si elle n'est pas considérée comme conforme, définissez ce membre dans une autre classe.
- Si ce membre doit rester non conforme, supprimez le mot clé **MustOverride** de sa définition, supprimez **CLSCompliantAttribute** de la définition de la classe ou marquez la classe comme `<CLSCompliant(False)>`.

Voir aussi

Référence

MustOverride

Concepts

Écriture d'un code conforme CLS

Le <NomMembre> non conforme CLS n'est pas autorisé dans une interface conforme CLS

Non-CLS-compliant <membername> is not allowed in a CLS-compliant interface

Une propriété, une procédure ou un événement dans une interface est marquée comme `<CLSCompliant(True)>` lorsque l'interface proprement dite est marquée comme `<CLSCompliant(False)>` ou n'est pas marquée.

Pour qu'une interface soit conforme au Spécification CLS (Common Language Specification) (CLS), tous ses membres doivent être conformes.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40033

Pour corriger cette erreur

- Si la conformité CLS est requise et si vous avez le contrôle du code source de l'interface, marquez l'interface comme `<CLSCompliant(True)>` si tous ses membres sont conformes.
- Si la conformité CLS est requise et si vous n'avez pas le contrôle du code source de l'interface ou si elle n'est pas considérée comme conforme, définissez ce membre dans une autre interface.
- Si ce membre doit rester dans son interface actuelle, supprimez **CLSCompliantAttribute** de sa définition ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Référence

Interface, instruction (Visual Basic)

Concepts

Écriture d'un code conforme CLS

L'objet créé à l'aide de Modifier & Continuer n'est pas encore disponible pour évaluation

Object created by Edit and Continue is not yet available for evaluation

Un nouvel objet est créé dans le débogueur Visual Studio, mais il n'est pas encore disponible.

Les causes possibles de l'indisponibilité d'un objet sont, entre autres, l'échec de l'allocation de mémoire et l'absence de synchronisation de l'état de l'objet. Ces conditions internes temporaires ne sont pas dues généralement à une entrée dans le débogueur.

ID d'erreur : BC30958

Pour corriger cette erreur

- Arrêtez et relancez le débogage.

Voir aussi

Tâches

Comment : démarrer l'exécution

Référence

Expressions dans Visual Basic

Concepts

Comment : arrêter le débogage ou l'exécution

Vue d'ensemble du code pas à pas

Autres ressources

Débogage dans Visual Studio

L'objet n'existe plus en raison d'une erreur de compilation ou d'une suppression

Object no longer exists due to compile error or deletion

Une erreur d'accès à un élément de code s'est produite dans le compilateur Visual Basic.

ID d'erreur : BC32300

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
3. Si l'erreur persiste après la réinstallation, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Les instructions 'On GoTo' et 'On GoSub' ne sont plus prises en charge

'On GoTo' and 'On GoSub' statements are no longer supported

L'utilisation de la valeur d'une variable ou d'une expression avec les instructions **GoTo** et **GoSub** pour contrôler le déroulement du programme n'est plus valide.

ID d'erreur : BC30817

Pour corriger cette erreur

- Restructurez votre application afin d'utiliser les instructions **If...Then...Else** ou **Case**.

Voir aussi

Référence

On Error, instruction (Visual Basic)
If...Then...Else, instruction (Visual Basic)
Case (Visual Basic)

Les instructions 'On Error' ne sont pas valides dans les instructions 'Using'

'On Error' statements are not valid within 'Using' statements

Une instruction **On Error** apparaît dans une instruction **Using**, mais n'est pas valide dans ce contexte.

ID d'erreur : BC36013

Pour corriger cette erreur

- Utilisez une gestion structurée des erreurs, par exemple un bloc **Try...Catch**, au lieu de l'instruction **On Error**

Voir aussi

Tâches

Comment : tester du code à l'aide d'un bloc Try...Catch dans Visual Basic

Référence

On Error, instruction (Visual Basic)

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Cas d'utilisation de la gestion structurée ou non structurée des exceptions

Les instructions 'On Error' ne sont pas valides dans les instructions 'SyncLock'

'On Error' statements are not valid within 'SyncLock' statements

Vous ne pouvez pas utiliser les instructions **On Error** dans des blocs **SyncLock** car elles interrompraient la synchronisation du thread.

ID d'erreur : BC30752

Pour corriger cette erreur

- Placez les instructions **On Error** à l'extérieur des blocs **SyncLock**.

Voir aussi

Référence

On Error, instruction (Visual Basic)

Les instructions 'On Error' ne sont pas valides dans la fenêtre Exécution

'On Error' statements are not valid in the Immediate window

Les instructions **On Error** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30720

Pour corriger cette erreur

- Supprimez les instructions **On Error** du code de débogage.

Voir aussi

Autres ressources

Débogage dans Visual Studio

L'argument omis ne peut pas correspondre à un paramètre ParamArray

Omitted argument cannot match a ParamArray parameter

Vous avez omis un argument requis pour correspondre à un paramètre **ParamArray**. **ParamArray**, qui vous permet de passer un nombre arbitraire d'arguments à une procédure, peut seulement être utilisé sur le dernier argument d'une liste d'arguments.

ID d'erreur : BC30588

Pour corriger cette erreur

- Spécifiez l'argument omis.

Voir aussi

Référence

ParamArray

'Of' requis lors de la spécification des arguments de type pour un type ou une méthode générique.

'Of' required when specifying type arguments for a generic type or method

Une instruction essaie de construire un type à partir d'un type générique, ou d'appeler une méthode générique, sans utiliser une clause Of.

La syntaxe Visual Basic pour les types génériques appelle les paramètres de type et les arguments de type à introduire par le mot clé **Of**. En outre, la liste des paramètres de type ou des arguments de type doit être mise entre parenthèses.

ID d'erreur : BC32093

Pour corriger cette erreur

- Insérez le mot clé **Of** au début de la liste des arguments de type et mettez la liste entière entre parenthèses.

Voir aussi

Tâches

Comment : utiliser une classe générique

Concepts

Types génériques en Visual Basic

Un seul bloc de commentaires XML est autorisé par élément de langage

Only one XML comment block is allowed per language element

Plusieurs blocs de commentaires XML ont été appliqués à un élément de langage.

ID d'erreur : BC42301

Pour corriger cette erreur

- Supprimez le bloc de commentaires XML superflu.

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Concepts

Documentation de votre code avec le langage XML (Visual Basic)

Vous pouvez spécifier uniquement une instance de type 'Public', 'Private', 'Protected', 'Friend' ou 'Protected Friend'

Only one of 'Public', 'Private', 'Protected', 'Friend', or 'Protected Friend' can be specified

Des conflits d'accessibilité se produisent dans une déclaration d'élément. Une seule accessibilité peut être spécifiée.

ID d'erreur : BC30176

Pour corriger cette erreur

- Choisissez une accessibilité et supprimez les autres mots clés de la déclaration.

Voir aussi

Concepts

Niveaux d'accès dans Visual Basic

Vous pouvez spécifier uniquement une instance de type 'NotOverridable', 'MustOverride' ou 'Overridable'

Only one of 'NotOverridable', 'MustOverride', or 'Overridable' can be specified

Des conflits de spécifications de substitution se produisent dans une déclaration d'élément. Les mots clés de substitution s'excluent mutuellement.

ID d'erreur : BC30177

Pour corriger cette erreur

- Choisissez un spécificateur de substitution et supprimez les autres mots clés de la déclaration.

Voir aussi

Concepts

Substitution de propriétés et de méthodes

Seuls les opérateurs de conversion peuvent être déclarés '<MotClé>'

Only conversion operators can be declared '<keyword>'

Une instruction Operator spécifie Widening ou Narrowing lorsque l'opérateur n'est pas un opérateur de conversion.

Chaque opérateur doit être déclaré comme Public (Visual Basic) et Shared (Visual Basic). Toutefois, seul un opérateur de conversion peut être déclaré avec Widening ou Narrowing, mais pas les deux.

Une définition d'opérateur peut contenir éventuellement les mots clés Overloads et Shadows. Aucun autre mot clé n'est autorisé dans une instruction Operator.

ID d'erreur : BC33019

Pour corriger cette erreur

- Supprimez le mot clé **Widening** ou **Narrowing** de la définition de l'opérateur. Celui-ci ne s'applique pas, car aucune conversion de type n'est en cours.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Autres ressources

Conversions de type en Visual Basic

L'opérateur n'est pas surchargeable

Operator is not overloadable

Seuls certains opérateurs prennent en charge la surcharge. Le tableau suivant présente les opérateurs que vous pouvez définir.

Type	Opérateurs
Unaire	**+**, **-**, **IsFalse**, **IsTrue**, **Not**
Binaire	**+**, **-**, *****, **/**, ****, **&**, **^**, **>>**, **<<**, **=**, **<>**, **>**, **>=**, **<**, **<=**, **And**, **Like**, **Mod**, **Or**, **Xor**
Conversion (unaire)	**CType**

Notez que l'opérateur **=** dans la liste binaire est l'opérateur de comparaison, et non l'opérateur d'assignation.

ID d'erreur : BC33002

Pour corriger cette erreur

1. Sélectionnez un opérateur dans le jeu d'opérateurs surchargeables.
2. Si vous avez besoin des fonctionnalités de surcharge d'un opérateur que vous ne pouvez pas surcharger directement, créez une procédure **Function** qui accepte les paramètres appropriés et retourne la valeur adéquate.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Function, instruction (Visual Basic)

Concepts

Procédures d'opérateur

La déclaration d'opérateur doit être l'une des suivantes : +,-,*,\,/,^, &, Like, Mod, And, Or, Xor, Not, <<, >>, =, <>, <, <=, >, >=, CType, IsTrue ou IsFalse

Operator declaration must be one of: +,-,*,\,/,^, &, Like, Mod, And, Or, Xor, Not, <<, >>, =, <>, <, <=, >, >=, CType, IsTrue, IsFalse

Vous pouvez déclarer uniquement un opérateur qui prend en charge la surcharge. Le tableau suivant présente les opérateurs que vous pouvez déclarer.

Type	**Opérateurs**
Unaire	**+**, **-**, **IsFalse**, **IsTrue**, **Not**
Binaire	**+**, **-**, *****, **/**, ****, **&**, **^**, **>>**, **<<**, **=**, **<>**, **>**, **>=**, **<**, **<=**, **And**, **Like**, **Mod**, **Or**, **Xor**
Conversion (unaire)	**CType**

Notez que l'opérateur **=** dans la liste binaire est l'opérateur de comparaison, et non l'opérateur d'assignation.

ID d'erreur : BC33000

Pour corriger cette erreur

1. Sélectionnez un opérateur dans le jeu d'opérateurs surchargeables.
2. Si vous avez besoin des fonctionnalités de surcharge d'un opérateur que vous ne pouvez pas surcharger directement, créez une procédure **Function** qui accepte les paramètres appropriés et retourne la valeur adéquate.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Function, instruction (Visual Basic)

Concepts

Procédures d'opérateur

L'opérateur '<nomopérateur>' n'est pas défini pour les types '<nomtype1>' et '<nomtype2>'

Message d'erreur
L'opérateur '<nomopérateur>' n'est pas défini pour les types '<nomtype1>' et '<nomtype2>'. Utilisez l'opérateur 'Is' pour comparer deux types référence.
Operator '<operatorname>' is not defined for types '<typename1>' and '<typename2>'. Use 'Is' operator to compare two reference types.

Une tentative d'utilisation d'un opérateur de manière inappropriée aux types spécifiés a été effectuée. Cette erreur peut être provoquée par l'utilisation de l'opérateur « = » au lieu de l'opérateur **Is** pour comparer deux objets.

ID d'erreur : BC31080

Pour corriger cette erreur

1. Utilisez l'opérateur **Is** pour comparer deux types référence.
2. Utilisez l'opérateur **Not** en association avec l'opérateur **Is** afin de montrer l'inégalité. Par exemple :

   VB

```
If Not A Is B Then
```

Voir aussi

Référence
Is, opérateur (Visual Basic)
=, opérateur (Visual Basic)
Not, opérateur (Visual Basic)

L'opérateur '<nomopérateur>' n'est pas défini pour les types '<type1>' et '<type2>'

Operator '<operatorname>' is not defined for types '<type1>' and '<type2>'

Vous avez tenté d'utiliser un opérateur binaire pour effectuer une opération non valide pour les deux éléments de code impliqués retournant une valeur.

ID d'erreur : BC30452

Pour corriger cette erreur

- Vérifiez les deux éléments et assurez-vous qu'ils sont compatibles.

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

Concepts

Opérateurs et expressions en Visual Basic

Association efficace d'opérateurs

L'opérateur '<nomopérateur>' n'est pas défini pour le type '<nomtype>'

Operator '<operatorname>' is not defined for type '<typename>'

Vous avez tenté d'utiliser un opérateur unaire (**+**, **-** ou **Not**) sur un type, tel que **String**, pour lequel il n'est pas défini.

ID d'erreur : BC30487

Pour corriger cette erreur

- Remplacez l'opérateur unaire par un opérateur approprié pour l'expression.

Voir aussi

Référence

Priorité des opérateurs en Visual Basic

Concepts

Opérateurs et expressions en Visual Basic

Association efficace d'opérateurs

L'opérateur '<opérateur>' doit avoir deux paramètres

Operator '<operator>' must have two parameters

Un opérateur binaire est défini avec moins ou plus de deux paramètres.

Un opérateur binaire doit contenir exactement deux paramètres.

ID d'erreur : BC33015

Pour corriger cette erreur

- Modifiez la définition pour spécifier exactement deux paramètres.
- Si un seul paramètre est requis, vous devez définir un opérateur unaire.
- Si aucun paramètre ou plus de deux paramètres sont requis, vous devez utiliser Function, instruction (Visual Basic) pour définir une procédure **Function** au lieu d'un opérateur.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

L'opérateur '<opérateur>' doit avoir un paramètre

Operator '<operator>' must have one parameter

Un opérateur unaire est défini sans paramètre ou avec plusieurs paramètres.

Un opérateur unaire doit avoir exactement un paramètre.

ID d'erreur : BC33014

Pour corriger cette erreur

- Modifiez la définition pour spécifier exactement un paramètre.
- Si deux paramètres sont requis, vous devez définir un opérateur binaire.
- Si aucun paramètre ou plus de deux paramètres sont requis, vous devez utiliser Function, instruction (Visual Basic) pour définir une procédure **Function** au lieu d'un opérateur.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

L'opérateur '<opérateur>' doit avoir un ou deux paramètres

Operator '<operator>' must have either one or two parameters

Un opérateur qui peut être unaire ou binaire est défini sans paramètres ou avec plus de deux paramètres.

Un opérateur unaire ou binaire doit avoir un ou deux paramètres.

ID d'erreur : BC33016

Pour corriger cette erreur

- Modifiez la définition pour spécifier un ou deux paramètres.
- Si aucun paramètre ou plus de deux paramètres sont requis, vous devez utiliser Function, instruction (Visual Basic) pour définir une procédure **Function** au lieu d'un opérateur.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

L'opérateur '<opérateur>' doit avoir un second paramètre de type 'Integer'

Operator '<operator>' must have a second parameter of type 'Integer'

Un opérateur de décalage de bits est déclaré avec le deuxième paramètre d'un type autre que **Integer**.

Lorsque vous utilisez l'opérateur de décalage vers la droite (**>>**) ou vers la gauche (**<<**) dans une expression, vous spécifiez le nombre de positions de décalage dans le second opérande. Pour cet opérande, Visual Basic vous permet de fournir un type de données qui s'étend à **Integer**. Toutefois, la définition du second opérande est strictement **Integer**. Si vous définissez une classe ou une structure avec un opérateur de décalage de bits sur cette classe ou structure, votre définition doit spécifier **Integer** pour le second opérande.

ID d'erreur : BC33041

Pour corriger cette erreur

- Modifiez la définition de l'opérateur de décalage de bits pour retourner une valeur **Integer**.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Opérateurs de décalage de bits

Concepts

Procédures d'opérateur

L'opérateur '<opérateur>' doit avoir un type de retour Boolean

Operator '<operator>' must have a return type of Boolean

Un opérateur de comparaison ou logique est déclaré avec un type de retour autre que Boolean, type de données (Visual Basic).

Le résultat d'un opérateur de comparaison (**=**, **<>**, **<**, **<=**, **>**, **>=**, **Is**, **IsNot**, **IsFalse**, **IsTrue**, **Like**, **TypeOf**) peut être uniquement **True** ou **False**. Pour plus d'informations, consultez Opérateurs de comparaison en Visual Basic.

Les opérateurs logiques (**And**, **AndAlso**, **Not**, **Or**, **OrElse**, **Xor**) fonctionnent intégralement dans le domaine de valeurs Boolean. Pour plus d'informations, consultez Opérateurs de bits et opérateurs logiques en Visual Basic.

ID d'erreur : BC33023

Pour corriger cette erreur

- Remplacez le type de retour de cet opérateur de comparaison ou logique par **Boolean**.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

L'opérateur '<SymboleOpérateur>' ne retourne pas une valeur pour tous les chemins de code

Message d'erreur
L'opérateur '<SymboleOpérateur>' ne retourne pas une valeur pour tous les chemins de code. Une exception de référence null peut se produire au moment de l'exécution lorsque le résultat est utilisé.
Operator '<operatorsymbol>' doesn't return a value on all code paths. A null reference exception could occur at run time when the result is used.

Une procédure d'opérateur contient au moins un chemin d'accès possible par l'intermédiaire de son code qui ne retourne pas de valeur.

Vous ne pouvez retourner la valeur d'une procédure d'opérateur qu'en l'incluant dans Return, instruction (Visual Basic).

Si le contrôle passe à l'instruction **End Operator**, la procédure d'opérateur retourne la valeur par défaut du type de données de la propriété. Pour plus d'informations, consultez « Comportement » dans Function, instruction (Visual Basic).

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42106

Pour corriger cette erreur

- Vérifiez votre logique de flux de contrôle et assurez-vous que chaque chemin d'accès possible se termine par une instruction **Return**. En particulier, la dernière instruction avant **End Operator** doit être une instruction **Return**.

Voir aussi

Référence
Operator, instruction
Concepts
Procédures d'opérateur

Opérandes de type Object utilisés dans les expressions pour les instructions 'Select', 'Case' ; des erreurs au moment de l'exécution peuvent se produire

Operands of type Object used in expressions for 'Select', 'Case' statements; runtime errors could occur

Une construction **Select**...**Case** utilise une ou plusieurs expressions dutype de données Object.

Lorsqu'une variable ou une expression a la valeur **Object**, le compilateur doit effectuer une *liaison tardive* qui provoque des opérations supplémentaires au moment de l'exécution. Il expose également votre application à des erreurs d'exécution potentielles. Par exemple, si vous assignez un Form à une variable **Object** et que vous essayez de le comparer à un nombre, le runtime lève une exception InvalidCastException parce que Visual Basic ne peut pas convertir un objet **Form** en une valeur numérique.

Les expressions d'une construction **Select**...**Case** doivent être toutes du même type de données ou étroitement liées à des types de données mutuellement convertibles. Il en est ainsi parce que chaque instruction **Case** compare au moins une valeur par rapport à l'expression test dont dépend la construction **Select**...**Case**.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42036

Pour corriger cette erreur

- Si possible, réorganisez toutes les expressions afin qu'elles aient pour valeurs des types de données pour lesquels les opérateurs de comparaison sont définis.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

Concepts

Opérateurs arithmétiques en Visual Basic

Opérateurs de comparaison en Visual Basic

Opérandes de type Object utilisés pour l'opérateur '<SymboleOpérateur>' ; des erreurs d'exécution peuvent se produire

Operands of type Object used for operator '<operatorsymbol>'; runtime errors could occur

Une expression utilise un opérateur pour lequel une ou les deux opérandes sont du type de données Object.

Lorsqu'une variable ou une expression a la valeur **Object**, le compilateur doit effectuer une *liaison tardive* qui provoque des opérations supplémentaires au moment de l'exécution. Il expose également votre application à des erreurs d'exécution potentielles. Par exemple, supposez que vous assignez un Form à une variable **Object** et que vous essayez ensuite de l'utiliser avec l'/, opérateur (Visual Basic). L'exécution lève ainsi une exception InvalidCastException parce que Visual Basic ne peut pas convertir un objet **Form** en une valeur numérique.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42019

Pour corriger cette erreur

- Si possible, réorganisez les opérandes afin qu'elles aient pour valeurs des types de données pour lesquels l'opérateur est défini.

Voir aussi

Concepts

Opérateurs arithmétiques en Visual Basic

Opérandes de type Object utilisés pour l'opérateur '<SymboleOpérateur>' ; utilisez l'opérateur 'IsNot' pour tester l'identité de l'objet.

Operands of type Object used for operator '<operatorsymbol>'; use the 'IsNot' operator to test object identity

Une expression utilise l'opérateur **<>** avec un ou les deux opérandes de type de données Object.

Vous devez utiliser l'opérateur **Is** ou **IsNot** pour déterminer si deux références d'objets font référence à la même instance d'objet. Consultez « Comparaison d'objets » dans Opérateurs de comparaison en Visual Basic.

Lorsqu'une variable ou une expression a la valeur **Object**, le compilateur doit effectuer une *liaison tardive* qui provoque des opérations supplémentaires au moment de l'exécution. Il expose également votre application à des erreurs d'exécution potentielles. Par exemple, si vous assignez un Form à une variable **Object** et que vous essayez de l'utiliser avec l'opérateur **<>**, l'exécution lève une exception InvalidCastException parce que Visual Basic ne peut pas convertir un objet **Form** en un type de données approprié pour la comparaison de valeurs. Même si les deux opérandes ont pour valeur le type **Form**, l'opération échoue car **<>** n'est pas défini pour les opérandes **Form**.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42032

Pour corriger cette erreur

- Si vous souhaitez déterminer si deux références d'objets font référence à la même instance d'objet, utilisez l'opérateur **Is** ou **IsNot**.

Voir aussi

Tâches

Comment : déterminer si deux objets sont liés

Comment : déterminer si deux objets sont identiques

Référence

IsNot, opérateur

Concepts

Opérateurs de comparaison en Visual Basic

Opérandes de type Object utilisés pour l'opérateur '<SymboleOpérateur>' ; utilisez l'opérateur 'Is' pour tester l'identité de l'objet

Operands of type Object used for operator '<operatorsymbol>'; use the 'Is' operator to test object identity

Une expression utilise **=** avec un ou les deux opérandes de type de données Object.

Vous devez utiliser l'opérateur **Is** ou **IsNot** pour déterminer si deux références d'objet font référence à la même instance d'objet. Consultez « Comparaison d'objets » dans Opérateurs de comparaison en Visual Basic.

Lorsqu'une variable ou une expression a la valeur **Object**, le compilateur doit exécuter une *liaison tardive* qui provoque des opérations supplémentaires au moment de l'exécution. Il expose également votre application à d'éventuelles erreurs d'exécution. Par exemple, si vous assignez Form à une variable **Object**, puis vous essayez de l'utiliser avec l'opérateur **=**, le runtime lève une exception InvalidCastException car Visual Basic ne peut pas convertir un objet **Form** en un type de données approprié pour la comparaison de valeurs. Même si les deux opérandes ont pour valeur le type **Form**, l'opération échoue car **=** n'est pas défini pour les opérandes **Form**.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42018

Pour corriger cette erreur

- Pour déterminer si deux références d'objet font référence à la même instance d'objet, utilisez l'opérateur **Is** ou **IsNot**.

Voir aussi

Tâches

Comment : déterminer si deux objets sont liés

Comment : déterminer si deux objets sont identiques

Référence

Is, opérateur (Visual Basic)

Concepts

Opérateurs de comparaison en Visual Basic

Les paramètres optionnels doivent spécifier une valeur par défaut

Optional parameters must specify a default value

Les paramètres optionnels doivent fournir des valeurs par défaut que vous pouvez utiliser si aucun paramètre n'est fourni par une procédure d'appel.

ID d'erreur : BC30812

Pour corriger cette erreur

- Spécifiez des valeurs par défaut pour les paramètres optionnels. Par exemple :

```
Sub Proc1(ByVal X As Integer, _
      Optional ByVal Y As String = "Default Value")
   MsgBox("Default argument is: " & Y)
End Sub
```

Voir aussi

Référence

Optional (Visual Basic)

Concepts

Déclaration de procédure pour les utilisateurs de Visual Basic 6.0

Les paramètres facultatifs ne peuvent pas avoir de types structure

Optional parameters cannot have structure types

Les paramètres facultatifs ne peuvent pas avoir de types structure.

ID d'erreur : BC31405

Pour corriger cette erreur

1. Déclarez des paramètres de types structure.
2. Déclarez un membre de classe statique pour remplacer la structure.

Voir aussi

Référence

Static (Visual Basic)

Autres ressources

Structures constituées à partir de vos propres types de données

Les paramètres optionnels ne peuvent pas être déclarés en tant que type '<type>'

Optional parameters cannot be declared as the type '<type>'

Les paramètres optionnels ne peuvent pas être du type de données d'une structure.

ID d'erreur : BC30423

Pour corriger cette erreur

1. Pour passer une structure à un paramètre optionnel, déclarez le paramètre en tant que **Object**.
2. Utilisez **CType** pour effectuer un cast du paramètre en ce type de structure dans la procédure.

Voir aussi

Référence

CType, fonction

Concepts

Structures et classes

'Optional' attendu

'Optional' expected

Un argument facultatif dans une déclaration de procédure est suivi d'un argument requis. Tous les arguments qui suivent un argument facultatif doivent également être facultatifs.

ID d'erreur : BC30202

Pour corriger cette erreur

1. Si l'argument doit être requis, déplacez-le afin qu'il précède le premier argument facultatif de la liste des arguments.
2. Si l'argument doit être facultatif, utilisez le mot clé **Optional**.

Voir aussi

Concepts

Paramètres facultatifs

Impossible de combiner 'Optional' et 'ParamArray'

'Optional' and 'ParamArray' cannot be combined

Les modificateurs **Optional** et **ParamArray** s'excluent mutuellement.

ID d'erreur : BC30642

Pour corriger cette erreur

- Spécifiez **ParamArray** ou **Optional**, mais pas les deux.

Voir aussi

Référence

ParamArray
Optional (Visual Basic)

Toutes les déclarations de variable de Option Strict On requièrent une clause 'As'

Option Strict On requires all variable declarations to have an 'As' clause

Une déclaration contient une variable déclarée sans clause **As**. Lorsque **Option Strict** a la valeur **On**, chaque variable, propriété, argument de procédure et retour de fonction doit être déclaré avec une clause **As** afin de spécifier son type de données, par exemple `Dim MyNum As Short`.

ID d'erreur : BC30209

Pour corriger cette erreur

1. Vérifiez si le mot clé **As** est mal orthographié.
2. Fournissez une clause **As** pour la variable déclarée ou définissez **Option Strict Off**.

Voir aussi

Référence

Option Strict, instruction

Concepts

Déclaration de variable dans Visual Basic

Toutes les déclarations Function, Property et Operator de Option Strict On requièrent une clause 'As'

Option Strict On requires all Function, Property, and Operator declarations to have an 'As' clause

Une déclaration contient une propriété déclarée ou un retour de fonction sans clause **As**. Lorsque **Option Strict** a la valeur **On**, chaque variable, propriété, argument de procédure et retour de fonction doit être déclaré avec une clause **As** afin de spécifier son type de données, par exemple `Dim MyNum As Short`.

ID d'erreur : BC30210

Pour corriger cette erreur

1. Vérifiez si le mot clé **As** est mal orthographié.
2. Fournissez une clause **As** pour la propriété déclarée ou le retour de fonction, ou définissez **Option Strict Off**.

Voir aussi

Référence

Option Strict, instruction

Concepts

Procédures Property

Function, procédures

Option Strict On interdit les opérandes de type Object pour l'opérateur '<nomopérateur>'

Option Strict On prohibits operands of type Object for operator '<operatorname>'

Les seuls opérateurs définis pour les variables objet sont **Is** et **TypeOf...Is**. Lorsque **Option Strict** a la valeur **On**, tous les opérandes doivent présenter les types de données définis pour l'opérateur donné.

ID d'erreur : BC30038

Pour corriger cette erreur

- Utilisez les fonctions de conversion de type appropriées, telles que **CInt** ou **CStr**, pour convertir les opérandes dans les types de données définis pour l'opérateur.

Voir aussi

Référence

Is, opérateur (Visual Basic)

Fonctions de conversion de types de données

Concepts

Opérateurs de comparaison en Visual Basic

Option Strict On n'autorise pas les opérandes de type Object pour l'opérateur '<nomopérateur>'

Message d'erreur
Option Strict On n'autorise pas les opérandes de type Object pour l'opérateur '<nomopérateur>'. Utilisez l'opérateur Is pour tester l'identité de l'objet.
Option Strict On disallows operands of type Object for operator '<operatorname>'. Use the Is operator to test for object identity.

Un opérateur de comparaison arithmétique, tel que le signe **=** est utilisé avec une ou plusieurs variables objet lorsque **Option Strict** a la valeur **On**.

ID d'erreur : BC32013

Pour corriger cette erreur

1. Utilisez **Option Strict Off** si les variables objet contiennent des valeurs numériques et que vous souhaitez effectuer une comparaison arithmétique.
2. Utilisez l'opérateur **Is** pour vérifier l'identité de l'objet par comparaison.

Voir aussi

Référence
Opérateurs de comparaison (Visual Basic)
Is, opérateur (Visual Basic)
Option Strict, instruction

Option Strict On interdit le passage du type '<nomtype1>' au type '<nomtype2>' lors de la recopie de la valeur du paramètre ByRef <nomparamètre>' dans l'argument correspondant

Option Strict On disallows narrowing from type '<typename1>' to type '<typename2>' in copying the value of ByRef parameter <parametername>' back to the matching argument

Un appel de procédure fournit un argument **ByRef** avec un type de données qui s'élargit au type déclaré de l'argument ; **Option Strict** a la valeur **On**. La conversion étendue est autorisée lorsque l'argument est passé à la procédure, mais lorsque la procédure modifie le contenu de l'argument variable dans le code appelant, la conversion inverse est restrictive. Les conversions restrictives ne sont pas autorisées avec **Option Strict On**.

ID d'erreur : BC32029

Pour corriger cette erreur

- Fournissez à chaque argument **ByRef** de l'appel de procédure le même type de données que le type déclaré ou définissez **Option Strict Off**.

Voir aussi

Référence

Option Strict, instruction

Concepts

Passage d'arguments par valeur et par référence
Conversions étendues et restrictives
Conversions implicites et explicites

Option Strict On rejette toute liaison tardive

Option Strict On disallows late binding

Visual Basic autorise des conversions implicites de n'importe quel type de données vers n'importe quel autre type de données. Toutefois, une perte de données peut survenir quand la valeur d'un type de données est convertie en un type de données avec moins de précision ou avec une capacité réduite. **Option Strict On** assure une notification au moment de la compilation de ces types de conversions afin qu'ils soient évités. Vous ne pouvez pas utiliser **Option Strict On** avec une liaison tardive.

ID d'erreur : BC30574

Pour corriger cette erreur

- Modifiez **Option Strict** en **Off**, en supprimant **On** ou en spécifiant explicitement **Off**.

Voir aussi

Référence

Option Strict, instruction

Concepts

Conversions étendues et restrictives

Option Strict On interdit les conversions implicites de '<type1>' en '<type2>' ; le type collection Visual Basic 6.0 n'est pas compatible avec le type collection .NET Framework.

Option Strict On interdit les conversions implicites de '`<type1>`' en '`<type2>`' ; le type collection Visual Basic 6.0 n'est pas compatible avec le type collection .NET Framework.

L'objet Collection utilisé dans Visual Basic 6.0 est différent de celui utilisé dans Visual Studio 2005.

ID d'erreur : BC30753

Pour corriger cette erreur

- Convertissez explicitement les objets de collection à l'aide de l'un des mots clés de conversion de type. Les mots clés CType Function et DirectCast lèvent une exception runtime si la conversion échoue Le mot clé TryCast retourne Nothing (Visual Basic) si la conversion échoue.

Voir aussi

Référence

CType, fonction
DirectCast
TryCast
Nothing (Visual Basic)

Concepts

Collections en Visual Basic

Option Strict On interdit les conversions implicites de '<type1>' en '<type2>'

Option Strict On disallows implicit conversions from '<type1>' to '<type2>'

Vous avez tenté de convertir un type en un autre type qui ne peut pas contenir la valeur, tel que la conversion d'un type **Long** en un type **Integer**, lorsque le commutateur de vérification de type (Option Strict, instruction) a la valeur **On**.

Ce type de conversion, appelé *conversion restrictive*, peut échouer au moment de l'exécution. Pour cette raison, **Option Strict On** interdit les conversions restrictives implicites.

ID d'erreur : BC30512

Pour corriger cette erreur

1. Déterminez si une conversion de n'importe quel type existe de `<type1>` à `<type2>`. S'ils sont tous deux des types élémentaires Visual Basic ou des instances de classes, vous pouvez généralement vous prononcer en consultant le tableau dans Conversions étendues et restrictives.
2. S'il n'existe qu'une conversion restrictive de `<type1>` à `<type2>`, vous devez utiliser un cast explicite. Les mots clés CType, fonction et DirectCast lèvent une exception runtime si la conversion échoue. Le mot clé TryCast s'applique uniquement aux types référence et retourne Nothing (Visual Basic) si la conversion échoue.
3. Si une conversion restrictive existe et que votre programme peut tolérer une défaillance au moment de l'exécution, ou si vous êtes certain qu'aucune défaillance au moment de l'exécution ne se produira, vous pouvez spécifier **Option Strict Off** au début de votre code source. Mais vous devez toujours englober la conversion dans un bloc Try...Catch...Finally, instruction (Visual Basic) pour éviter des résultats inattendus ou la fin anticipée de votre programme.
4. Si aucune conversion n'existe de `<type1>` à `<type2>`, vous devez réévaluer votre logique de programme. Vous pouvez être en mesure d'écrire du code qui assigne des valeurs à `<type2>` correspondant aux valeurs anticipées de `<type1>`.
5. Si aucune conversion n'existe de `<type1>` à `<type2>` et que l'un des types est une classe ou une structure que vous avez définie, vous pouvez être en mesure de définir un opérateur de conversion de ce type à l'autre type ou de l'autre type. Pour plus d'informations, consultez Comment : définir un opérateur de conversion.
6. Dans tous les cas et en règle générale, vous devez éviter d'utiliser des conversions restrictives, sauf si vous pouvez intercepter les défaillances dans un bloc **Catch** et les traiter efficacement.

Voir aussi

Tâches

Comment : définir un opérateur de conversion

Référence

Option Strict, instruction
CType, fonction
DirectCast
TryCast
Nothing (Visual Basic)
Try...Catch...Finally, instruction (Visual Basic)

Concepts

Conversions étendues et restrictives

Option Strict Custom ne peut être utilisé qu'en tant qu'option du compilateur de ligne de commande (vbc.exe)

Option Strict Custom can only be used as an option to the command-line compiler (vbc.exe)

L'instruction **Option Strict** prend uniquement **On** et **Off** comme arguments ; `Option Strict Custom` n'est pas autorisé.

Utilisez l'option du compilateur **/optionstrict:custom** à avertir lorsque la syntaxe de langue stricte n'est pas respectée.

ID d'erreur : BC31141

Pour corriger cette erreur

1. Supprimez `Option Strict Custom` du code source.
2. Spécifiez l'option **/optionstrict:custom**. Pour plus d'informations, consultez /optionstrict.

Voir aussi

Référence

Option (Visual Basic)
Option Strict, instruction
/optionstrict

'Option Strict' ne peut être suivi que de 'On' ou 'Off'

'Option Strict' can be followed only by 'On' or 'Off'

Les instructions **Option Strict** doivent spécifier **On** ou **Off**. Toute autre valeur est non valide.

ID d'erreur : BC30620

Pour corriger cette erreur

- Spécifiez **On** ou **Off** pour l'instruction **Option Strict**.

Voir aussi

Référence

Option (Visual Basic)
Option Compare, instruction
Option Explicit, instruction (Visual Basic)
Option Strict, instruction

Toutes les déclarations et instructions 'Imports' doivent être précédées d'instructions 'Option'

'Option' statements must precede any declarations or 'Imports' statements

Les instructions **Option** doivent être les premières instructions non commentées de votre code source.

ID d'erreur : BC30627

Pour corriger cette erreur

- Déplacez les instructions **Option** au début de votre code source, juste avant les instructions **Imports** et **Namespace**.

Voir aussi

Référence

Option (Visual Basic)
Option Compare, instruction
Option Explicit, instruction (Visual Basic)
Option Strict, instruction

'Option' doit être suivi de 'Compare', 'Explicit' ou 'Strict'

'Option' must be followed by 'Compare', 'Explicit', or 'Strict'

Les seules options que vous pouvez définir avec l'instruction **Option** sont **Compare**, **Explicit** et **Strict**.

ID d'erreur : BC30206

Pour corriger cette erreur

1. Vérifiez si un mot clé d'option est mal orthographié.
2. Vérifiez si l'instruction **Option** est prématurément terminée.

Voir aussi

Référence

Option (Visual Basic)
Option Compare, instruction
Option Explicit, instruction (Visual Basic)
Option Strict, instruction

'Option Explicit' ne peut être suivi que de 'On' ou 'Off'

'Option Explicit' can be followed only by 'On' or 'Off'

Les instructions **Option Explicit** doivent spécifier **On** ou **Off**. Toute autre valeur est non valide.

ID d'erreur : BC30640

Pour corriger cette erreur

- Spécifiez **Option Explicit On** ou **Option Explicit Off**.

Voir aussi

Référence

Option Explicit, instruction (Visual Basic)

'Option Compare' doit être suivi de 'Text' ou 'Binary'

'Option Compare' must be followed by 'Text' or 'Binary'

Une instruction **Option Compare** contient un paramètre incorrect ou aucun paramètre. Les seuls paramètres autorisés dans **Option Compare** sont **Text** et **Binary**.

ID d'erreur : BC30207

Pour corriger cette erreur

1. Vérifiez si le spécificateur de paramètre est mal orthographié.
2. Ajoutez **Text** ou **Binary** à l'instruction **Option Compare**, par exemple `Option Compare Text`.

Voir aussi

Référence

Option Compare, instruction

Une instruction 'Option <spécificateur>' ne peut apparaître qu'une seule fois par fichier

'Option <specifier>' statement can only appear once per file

Les instructions **Option Compare**, **Option Explicit** et **Option Strict** ne peuvent être utilisées qu'une seule fois chacune dans le même fichier source.

ID d'erreur : BC30225

Pour corriger cette erreur

- Supprimez l'instruction **Option** dupliquée.

Voir aussi

Référence

Option (Visual Basic)
Option Compare, instruction
Option Explicit, instruction (Visual Basic)
Option Strict, instruction

L'option <nomoption> requiert <argument>

Option <optionname> requires <argument>

Un argument requis a été omis.

ID d'erreur : BC2006

Pour corriger cette erreur

- Spécifiez l'argument manquant.

Voir aussi

Concepts

Types d'erreurs

l'option <nomoption> ne peut être suivie que de '+' ou '-'

option <optionname> can be followed only by '+' or '-'

Le texte qui suit l'option n'est ni un signe plus (+) ni un signe moins (-).

ID d'erreur : BC2009

Pour corriger cette erreur

- Remplacez le texte qui suit l'option avec un signe plus (+) pour activer l'option.

  - ou -

- Remplacez le texte qui suit l'option avec un signe moins (-) pour désactiver l'option.

Voir aussi

Référence

Options du compilateur Visual Basic par ordre alphabétique

Concepts

Types d'erreurs

Les opérateurs doivent être déclarés 'Shared'

Operators must be declared 'Shared'

Une instruction Operator ne contient pas le mot clé Shared (Visual Basic).

Une procédure **Operator** requiert les mots clés Public (Visual Basic) et **Shared**, et un opérateur de conversion requiert également le mot clé Widening ou Narrowing.

ID d'erreur : BC33012

Pour corriger cette erreur

- Insérez le mot clé **Shared** dans l'instruction **Operator**.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs doivent être déclarés 'Public'

Operators must be declared 'Public'

Une instruction Operator ne contient pas le mot clé Public (Visual Basic).

Une procédure **Operator** requiert les mots clés **Public** et Shared (Visual Basic), et un opérateur de conversion requiert également le mot clé Widening ou Narrowing.

ID d'erreur : BC33011

Pour corriger cette erreur

- Insérez le mot clé **Public** dans l'instruction **Operator**.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les opérateurs ne peuvent pas être déclarés dans des modules

Operators cannot be declared in Modules

Une instruction Operator apparaît dans une définition de module.

Vous pouvez définir un opérateur dans le cadre d'une classe ou d'une structure que vous définissez. Cet opérateur doit accepter cette classe ou structure comme au moins l'un de ses opérandes.

Un opérateur doit utiliser une instance d'un élément de programmation comme l'un de ses opérandes, et seules les classes et les structures contiennent des instances. Par conséquent, vous ne pouvez pas définir un opérateur dans le cadre de tout autre élément de programmation.

ID d'erreur : BC33018

Pour corriger cette erreur

- Si une opération est requise sur le module, utilisez Function, instruction (Visual Basic) pour définir une procédure **Function** qui exécute l'opération.
- Vous pouvez également définir une classe ou une structure dans le module et définir un opérateur sur cette classe ou structure. Toutefois, cet opérateur doit accepter une instance de cette classe ou structure comme au moins l'un de ses opérandes.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Concepts

Procédures d'opérateur

Les opérateurs ne peuvent pas être déclarés '<MotClé>'

Operators cannot be declared '<keyword>'

Un opérateur est déclaré avec un mot clé incorrect pour les définitions de l'opérateur.

Chaque opérateur doit être déclaré comme Public (Visual Basic) et Shared (Visual Basic). En outre, un opérateur de conversion doit être déclaré avec Widening ou Narrowing et non les deux à la fois. Une définition de l'opérateur peut éventuellement inclure les mots clés Overloads ou Shadows. Aucun autre mot clé n'est autorisé dans une instruction Operator.

ID d'erreur : BC33013

Pour corriger cette erreur

- Supprimez le mot clé incorrect de la définition de l'opérateur.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Concepts

Procédures d'opérateur

Opérateur sans clause 'As' ; type Object pris par défaut

Operator without an 'As' clause; type of Object assumed

Une procédure d'opérateur ne spécifie pas de clause **As**.

Une clause **As** identifie un type de données à associer à un élément de programmation. Dans une instruction Operator, elle spécifie le type de données de la valeur retournée par la procédure d'opérateur au code appelant. Si vous n'incluez pas une clause **As** dans l'instruction **Operator**, le type de données retournées a la valeur **Object** par défaut.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41005

Pour corriger cette erreur

- Incluez une clause **As** dans l'instruction **Operator** pour spécifier le type de données retournées.

Voir aussi

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Mémoire insuffisante (erreur du compilateur Visual Basic)

Out of memory (Visual Basic Compiler Error)

Une mémoire supérieure à celle disponible est requise.

ID d'erreur : BC2004

Pour corriger cette erreur

- Fermez les applications, les documents et les fichiers source dont vous n'avez pas besoin.
- Éliminez les contrôles et les formulaires dont vous n'avez pas besoin afin d'en charger moins à la fois.
- Réduisez le nombre de variables **Public**.
- Vérifiez l'espace disque disponible.
- Augmentez la mémoire vive disponible en installant plus de mémoire ou en la réallouant.
- Assurez-vous que cette mémoire est libérée lorsqu'elle n'est plus nécessaire.

Voir aussi

Concepts

Types d'erreurs

Dépassement de capacité (erreur Visual Basic)

Overflow (Visual Basic Error)

Un littéral représente une valeur en dehors des limites du type de données auquel il est assigné.

ID d'erreur : BC30036

Pour corriger cette erreur

- Consultez l'intervalle correspondant au type de données cible et réécrivez le littéral afin qu'il soit conforme à cet intervalle.

Voir aussi

Référence

Liste des types de données (Visual Basic)

La résolution de surcharge a échoué, car aucune '<méthode>' n'est accessible

Overload resolution failed because no '<method>' is accessible

Vous avez effectué un appel vers une méthode surchargée, mais le compilateur ne peut pas trouver de méthode accessible avec une liste d'arguments correspondant au nombre de paramètres.

ID d'erreur : BC30517

Pour corriger cette erreur

1. Vérifiez la liste d'arguments fournie.
2. Vérifiez la méthode que vous essayez d'appeler afin de déterminer si elle est accessible.

Voir aussi

Concepts

Propriétés et méthodes surchargées

La résolution de surcharge a échoué, car aucun '<NomProcédureGénérique>' accessible n'accepte ce nombre d'arguments de type

Overload resolution failed because no accessible '<genericprocedurename>' accepts this number of type arguments

Un appel à une procédure générique surchargée ne peut pas être résolu car le compilateur ne peut pas accéder à une version surchargée avec le nombre approprié de paramètres de type.

Lorsque vous appelez une procédure générique, vous devez fournir un argument de type pour chaque paramètre de type. Vous pouvez également ne pas fournir d'arguments de type et laisser le compilateur effectuer une *inférence de type*. Pour plus d'informations, consultez « Inférence du type » dans Procédures génériques dans Visual Basic.

ID d'erreur : BC32087

Pour corriger cette erreur

1. Vérifiez que la version que vous comptez appeler est accessible par le code appelant. Consultez Niveaux d'accès dans Visual Basic.
2. Ajoutez ou supprimez les arguments de type de votre code appelant afin que la liste des arguments de type corresponde à la liste des paramètres de type de la version que vous comptez appeler.

   - ou -

   Supprimez tous les arguments de type de votre code appelant et laissez le compilateur effectuer une inférence de type. N'oubliez pas que l'inférence de type peut échouer en cas de conflits ou d'ambiguïtés.

Voir aussi

Référence

Liste de types

Concepts

Propriétés et méthodes surchargées
Résolution de surcharge
Types génériques en Visual Basic

La résolution de surcharge a échoué, car aucune '<méthode>' accessible n'accepte ce nombre d'arguments

Overload resolution failed because no accessible '<method>' accepts this number of arguments

Vous avez effectué un appel vers une méthode surchargée, mais le compilateur ne peut pas trouver de méthode avec une liste d'arguments correspondante.

ID d'erreur : BC30516

Pour corriger cette erreur

- Vérifiez la liste d'arguments fournie.

Voir aussi

Concepts

Propriétés et méthodes surchargées

La résolution de surcharge a échoué, car aucune '<méthode>' accessible ne peut être appelée avec ces arguments : <erreur>

Overload resolution failed because no accessible '<method>' can be called with these arguments:<error>

Vous avez effectué un appel vers une méthode surchargée, mais le compilateur ne peut pas trouver de méthode avec une liste d'arguments correspondante.

ID d'erreur : BC30518

Pour corriger cette erreur

- Vérifiez la liste d'arguments fournie.

Voir aussi

Concepts

Propriétés et méthodes surchargées

La résolution de surcharge a échoué, car aucune '<méthode>' accessible ne peut être appelée sans conversion restrictive : <erreur>

Overload resolution failed because no accessible '<method>' can be called without a narrowing conversion: <error>

Vous avez effectué un appel vers une méthode surchargée, mais le compilateur ne peut pas trouver de méthode pouvant être appelée sans conversion restrictive. Dans une conversion restrictive, la valeur est convertie dans un type de données susceptible de ne pas pouvoir accueillir avec précision certaines valeurs possibles des données.

ID d'erreur : BC30519

Pour corriger cette erreur

- Spécifiez **Option Strict Off**.

Voir aussi

Référence

Option Strict, instruction

Concepts

Propriétés et méthodes surchargées

Conversions étendues et restrictives

La résolution de surcharge a échoué, car aucune '<méthode>' accessible n'est plus spécifique pour ces arguments : <erreur>

Vous avez effectué un appel vers une méthode surchargée, mais le compilateur a trouvé au moins deux surcharges avec des listes de paramètres vers lesquelles votre liste d'arguments peut être convertie ; en conséquence, il ne peut pas effectuer de sélection parmi celles-ci.

Le compilateur tente de faire correspondre les types de données dans la liste d'arguments appelante et la liste de paramètres de surcharge aussi étroitement que possible. Il nécessite une conversion étendue de chacun de vos arguments en son paramètre correspondant, que le commutateur de vérification de type (instruction Option Strict) soit **On** ou **Off**.

Si le compilateur trouve plusieurs surcharges qui satisfont au critère d'extension, il recherche ensuite la surcharge qui est la plus spécifique pour les types de données d'argument, c'est-à-dire celle qui appelle l'extension la moins importante. Il génère ce message d'erreur lorsqu'une surcharge est plus spécifique pour le type de données d'un argument alors qu'une autre surcharge est plus spécifique pour le type de données d'un autre argument. Pour obtenir des informations supplémentaires et un exemple, consultez Résolution de surcharge.

ID d'erreur : BC30521

Pour corriger cette erreur

1. Examinez toutes les surcharges pour la méthode et déterminez celle que vous souhaitez appeler.
2. Dans votre instruction appelante, faites correspondre les types de données des arguments aux types de données des paramètres définis pour la surcharge souhaitée. Vous devrez peut-être utiliser CType Function pour convertir un ou plusieurs types de données en types définis.

Voir aussi

Référence

Overloads
Option Strict, instruction
CType, fonction

Concepts

Surcharge de procédure
Considérations sur les surcharges de procédures
Résolution de surcharge
Propriétés et méthodes surchargées

La surcharge de méthodes déclarées dans plusieurs interfaces de base n'est pas valide

Overloading methods declared in multiple base interfaces is not valid

Plusieurs interfaces héritées surchargent implicitement la même méthode.

ID d'erreur : BC31410

Pour corriger cette erreur

- Utilisez le modificateur **Shadows** à la place du modificateur **Overloads**.

Voir aussi

Référence

Overloads
Shadows

Le paramètre ParamArray doit être un tableau unidimensionnel

ParamArray parameter must be a one-dimensional array

Un argument **ParamArray** doit être déclaré en tant que type tableau unidimensionnel. La déclaration spécifie plusieurs dimensions pour l'argument **ParamArray**.

ID d'erreur : BC30051

Pour corriger cette erreur

- Supprimez toutes les virgules à l'intérieur des parenthèses après le nom d'argument **ParamArray**.

Voir aussi

Référence

ParamArray

Concepts

Tableaux de paramètres

Autres ressources

Tableaux dans Visual Basic

Le paramètre ParamArray doit être un tableau

ParamArray parameter must be an array

Un argument **ParamArray** doit être déclaré en tant que type tableau unidimensionnel. La déclaration n'inclut pas les parenthèses après le nom d'argument **ParamArray**.

ID d'erreur : BC30050

Pour corriger cette erreur

- Ajoutez une paire de parenthèses après le nom d'argument **ParamArray**.

Voir aussi

Référence

ParamArray

Concepts

Tableaux de paramètres

Autres ressources

Tableaux dans Visual Basic

Les paramètres ParamArray doivent être déclarés 'ByVal'

ParamArray parameters must be declared 'ByVal'

Les paramètres **ParamArray** ne peuvent pas utiliser le modificateur **ByRef**.

ID d'erreur : BC30667

Pour corriger cette erreur

- Déclarez les paramètres **ParamArray** à l'aide du modificateur **ByVal**.

Voir aussi

Référence

ParamArray
ByRef
ByVal

Les paramètres ParamArray doivent avoir un type tableau

ParamArray parameters must have an array type

Un problème est survenu lors de l'accès à l'assembly MSCorlib, ce qui a entraîné la non-validité du type des paramètres **ParamArray**.

ID d'erreur : BC31092

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si l'erreur se reproduit, enregistrez votre travail et redémarrez Visual Studio.
3. Si l'erreur persiste, réinstallez Visual Basic.
4. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Référence

ParamArray

Autres ressources

Support technique et accessibilité

Un argument omis correspond déjà au paramètre '<nomparamètre>' dans '<nomméthode>'

Parameter '<parametername>' in '<methodname>' already has a matching omitted argument

Un appel de procédure fournit un argument par nom après omission du même argument par position ; par exemple :

```
Public Sub ABC(ByVal X As Byte, Optional ByVal Y As Byte = 0, _
                                Optional ByVal Z As Byte = 0)
' ...
Call ABC(6, , Y:=3)   ' Argument Y omitted by position, supplied by name.
```

ID d'erreur : BC32021

Pour corriger cette erreur

- Fournissez l'argument par position ou supprimez la virgule qui provoque son omission.

Voir aussi

Concepts

Passage des arguments par position et par nom

Le paramètre '<nomparamètre>' de '<nomméthode>' a déjà un argument correspondant

Parameter '<parametername>' of '<methodname>' already has a matching argument

Un argument est passé par nom plusieurs fois dans le même appel de procédure.

ID d'erreur : BC30274

Pour corriger cette erreur

- Vérifiez si l'un des noms d'arguments est mal orthographié.
- Supprimez l'argument dupliqué de la liste d'arguments.

Voir aussi

Concepts

Passage des arguments par position et par nom

Paramètre déjà déclaré avec le nom '<nomargument>'

Parameter already declared with name '<argumentname>'

Une déclaration de procédure définit deux arguments avec le même nom. Tous les noms d'arguments doivent être uniques au sein d'une procédure de déclaration.

ID d'erreur : BC30237

Pour corriger cette erreur

- Changez le nom de l'un des arguments.

Voir aussi

Concepts

Paramètres et arguments d'une procédure
Résolution d'une référence lorsque plusieurs variables ont le même nom

Un paramètre ne peut pas avoir le même nom que sa fonction de définition

Parameter cannot have the same name as its defining function

Vous avez utilisé le même nom pour un paramètre et sa fonction de définition.

ID d'erreur : BC30530

Pour corriger cette erreur

- Vérifiez les noms de paramètres dans la fonction et modifiez le nom de paramètre en conflit.

Voir aussi

Référence

Éditeur de collection Parameters

Le spécificateur de paramètre est dupliqué

Parameter specifier is duplicated

Un spécificateur de paramètre est appliqué plusieurs fois au paramètre.

ID d'erreur : BC30785

Pour corriger cette erreur

- Supprimez les spécificateurs dupliqués.

Voir aussi

Référence

Liste de paramètres

Le paramètre de cet opérateur unaire doit être du type conteneur '<NomType>'

Parameter type of this unary operator must be the containing type '<typename>'

Une définition d'un opérateur unaire spécifie un paramètre avec un type autre que celui de la classe ou la structure dans laquelle l'opérateur est défini.

Lorsque vous définissez un opérateur dans une classe ou une structure, au moins l'un des paramètres doit être du type de cette classe ou structure. Dans le cas d'un opérateur unaire, l'opérande unique doit être du type de cette classe ou structure.

ID d'erreur : BC33020

Pour corriger cette erreur

- Remplacez le type de paramètre par le type de la classe ou la structure dans laquelle l'opérateur est défini.
- Si vous souhaitez utiliser un type de données comme paramètre et retourner un autre type de données comme résultat de l'opération, définissez plutôt un opérateur de conversion.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les types de paramètres de '<opérateur>' doivent être '<NomType>' pour être utilisés dans une instruction 'For'

Parameter types of '<operator>' must be '<typename>' to be used in a 'For' statement

Une boucle **For** spécifie une variable compteur d'un type qui ne définit pas l'opérateur **>=** ou **<=** avec les paramètres de son propre type.

La variable compteur doit être d'un type qui prend en charge les opérateurs supérieur ou égal (**>=**) et inférieur ou égal (**<=**) qui comparent leur type conteneur. Cela signifie que les deux opérandes doivent être du type de variable compteur.

Si vous utilisez un type de données numériques pour la variable compteur, les opérateurs **>=** et **<=** sont pris en charge sur le type conteneur. Si vous utilisez une classe ou une structure définie par l'utilisateur, vous devez définir les deux opérateurs avec les opérandes du type de votre classe ou structure.

ID d'erreur : BC33040

Pour corriger cette erreur

1. Vérifiez que l'orthographe du type de données de la variable compteur est correcte.
2. Si vous utilisez une classe ou une structure définie par l'utilisateur pour la variable compteur, définissez les opérateurs **>=** et **<=** qui comparent cette classe ou structure.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

For...Next, instruction (Visual Basic)
Operator, instruction

Concepts

Procédures d'opérateur

Problème possible détecté pendant la génération de l'assembly : <erreur>

Possible problem detected while building assembly: <error>

L'outil ALink, appelé par le compilateur Visual Basic, signale une erreur lors de la génération de l'assembly. L'une des causes possibles est un assembly signé faisant référence à un assembly non signé.

Ce message est un avertissement. Le compilateur poursuit la génération de l'assembly. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40009

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Recompilez le programme afin de vérifier si l'erreur se reproduit.
3. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
4. Si l'erreur persiste après la réinstallation, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Problème possible détecté pendant la génération de l'assembly '<nomassembly>' : <erreur>

Possible problem detected while building assembly '<assemblyname>': <error>

L'outil ALink, appelé par le compilateur Visual Basic, signale une erreur lors de la génération de l'assembly. Les causes de cette erreur peuvent être les suivantes :

- Un assembly signé faisant référence à un assembly non signé. Dans ce cas, vous devez déterminer si l'assembly référencé répond à vos critères de sécurité.
- Génération d'une application 64 bits sur une plate-forme 32 bits. Dans ce cas, vous devez vous assurer que les versions 64 bits de tous les assemblys référencés sont installées sur la plate-forme cible. Pour un assembly Common Language Runtime (CLR), cela est géré automatiquement, bien que ce message d'erreur apparaisse toujours.

Ce message est un avertissement. Le compilateur poursuit la génération de l'assembly. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40010

Pour corriger cette erreur

1. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.
2. Recompilez le programme afin de vérifier si l'erreur se reproduit.
3. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
4. Si l'erreur persiste après la réinstallation, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Concepts

Vue d'ensemble du Common Language Runtime

Autres ressources

Support technique et accessibilité

La propriété '<nompropriété>' ne peut pas être déclarée 'Default' car elle se substitue à une propriété dans la base <type> '<nomtype>' qui n'est pas une valeur par défaut

Property '<propertyname>' cannot be declared 'Default' because it overrides a Property on the base <type> '<typename>' that is not default

Vous avez tenté de déclarer une propriété **Default**, ce qui n'est pas une déclaration valide car elle substitue une propriété qui n'a pas été déclarée **Default** dans la classe de base.

ID d'erreur : BC30504

Pour corriger cette erreur

- Supprimez la déclaration **Default**.

Voir aussi

Référence

Default (Visual Basic)

La propriété '<NomPropriété>' ne retourne pas une valeur pour tous les chemins de code

Message d'erreur
La propriété '<NomPropriété>' ne retourne pas une valeur pour tous les chemins de code. Une exception de référence null peut se produire au moment de l'exécution lorsque le résultat est utilisé.
Property '<propertyname>' doesn't return a value on all code paths. A null reference exception could occur at run time when the result is used.

Une procédure **Get** de propriété contient au moins un chemin d'accès possible par l'intermédiaire de son code qui ne retourne pas de valeur.

Vous pouvez retourner la valeur d'une procédure **Get** de propriété de plusieurs façons :

- Assignez la valeur au nom de la propriété, puis exécutez une instruction **Exit Property**.
- Assignez la valeur au nom de la propriété, puis exécutez l'instruction **End Get**.
- Incluez la valeur dans Return, instruction (Visual Basic).

Si le contrôle passe sur **Exit Property** ou **End Get** et si vous n'avez pas assigné une valeur au nom de la propriété, la procédure **Get** retourne la valeur par défaut du type de données de la propriété. Pour plus d'informations, consultez « Comportement » dans Function, instruction (Visual Basic).

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42107

Pour corriger cette erreur

- Vérifiez votre logique de flux de contrôle et veillez à assigner une valeur avant chaque instruction qui provoque un renvoi.

  Il est plus facile de vérifier que chaque renvoi de la procédure retourne une valeur si vous utilisez toujours l'instruction **Return**. En procédant ainsi, la dernière instruction avant **End Get** doit correspondre à une instruction **Return**.

Voir aussi

Référence
Property, instruction
Get, instruction
Concepts
Procédures Property

Les propriétés sans paramètre requis ne peuvent pas être déclarées 'Default'

Properties with no required parameters cannot be declared 'Default'

Une propriété a été spécifiée sans paramètres requis, mais avec le modificateur **Default**.

ID d'erreur : BC31048

Pour corriger cette erreur

- Déclarez des paramètres requis pour les propriétés par défaut.
- Supprimez le modificateur **Default**de la définition de propriétés.

Voir aussi

Concepts

Propriétés par défaut

Les propriétés d'un module ne peuvent pas être déclarées '<spécificateur>'

Properties in a Module cannot be declared '<specifier>'

Vous avez tenté d'utiliser un spécificateur, tel que **MustOverride**, **NotOverridable** ou **Overridable**, sur une propriété dans un module.

ID d'erreur : BC30503

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Module, instruction
Module (Visual Basic)

Les propriétés déclarées 'WriteOnly' ne peuvent pas avoir de 'Get'

Properties declared 'WriteOnly' cannot have a 'Get'

La procédure **Get** lit la valeur d'une propriété. Les propriétés **WriteOnly** ne peuvent pas être lues.

ID d'erreur : BC30023

Pour corriger cette erreur

- Supprimez le mot clé **WriteOnly** de l'instruction **Property** ou la procédure **Get** du corps de la propriété.

Voir aussi

Référence

Property, instruction
Get, instruction
WriteOnly

Les propriétés déclarées 'ReadOnly' ne peuvent pas avoir de 'Set'

Properties declared 'ReadOnly' cannot have a 'Set'

La procédure **Set** écrit la valeur d'une propriété. Les propriétés **ReadOnly** ne peuvent pas être écrites.

ID d'erreur : BC30022

Pour corriger cette erreur

- Supprimez le mot clé **ReadOnly** de l'instruction **Property** ou la procédure **Set** du corps de la propriété.

Voir aussi

Référence

Property, instruction
Set, instruction (Visual Basic)
ReadOnly (Visual Basic)

Les propriétés ne peuvent pas être déclarées'<nommodificateur>'

Properties cannot be declared '<modifiername>'

Il existe une erreur dans la syntaxe utilisée pour la déclaration **Property**. Cette erreur indique généralement qu'un mot clé spécifié, tel que **WithEvents**, ne peut pas être utilisé dans une déclaration **Property**.

ID d'erreur : BC30639

Pour corriger cette erreur

- Vérifiez la syntaxe de la déclaration **Property**.

Voir aussi

Concepts

Procédures Property

Constante de compilation conditionnelle au niveau du projet non valide : <erreur>

Project-level conditional compilation constant is not valid: <error>

Une option de compilation conditionnelle non valide est spécifiée pour le projet.

ID d'erreur : BC31031

Pour corriger cette erreur

- Vérifiez les constantes de compilation conditionnelle utilisées avec ce projet.

Voir aussi

Autres ressources

Compilation conditionnelle (Visual Basic)

Constante de compilation conditionnelle au niveau du projet '<nomconstante>' non valide : <erreur>

Project-level conditional compilation constant '<constantname>' is not valid: <error>

Une option de compilation conditionnelle non valide est spécifiée pour le projet.

ID d'erreur : BC31030

Pour corriger cette erreur

- Vérifiez les constantes de compilation conditionnelle utilisées avec ce projet.

Voir aussi

Autres ressources

Compilation conditionnelle (Visual Basic)

Le projet a été fermé

Project has been closed

Une erreur d'accès à un modèle de code s'est produite dans le compilateur Visual Basic.

ID d'erreur : BC32301

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
3. Si l'erreur persiste après la réinstallation, rassemblez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Le projet a déjà une référence à l'assembly <IdentitéAssembly>

Message d'erreur
Le projet a déjà une référence à l'assembly <IdentitéAssembly>. Une seconde référence à '<CheminFichier>' ne peut pas être ajoutée.
Project already has a reference to assembly <assemblyidentity>. A second reference to '<filepath>' cannot be added.

Un projet crée plusieurs références au même assembly.

L'identité de l'assembly inclut son nom, sa version, sa clé publique, si elle existe, et sa culture.

Une cause possible de cette erreur est une référence à une autre copie de l'assembly via un chemin d'accès différent de celui de la référence d'origine.

ID d'erreur : BC32208

Pour corriger cette erreur

- Supprimez la seconde référence. Elle est inutile car elle fait référence au même assembly.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Dépannage de références rompues
Concepts
Références de projet

Le projet '<NomProjet1>' établit une référence indirecte au projet '<NomProjet2>', qui contient '<NomType>'

Message d'erreur
Le projet '<NomProjet1>' établit une référence indirecte au projet '<NomProjet2>', qui contient '<NomType>'. Ajoutez une référence de projet pour '<NomProjet2 >' dans votre projet.
Project '<projectname1>' makes an indirect reference to project '<projectname2>', which contains '<typename>'. Add a project reference to '<projectname2>' to your project.

Le code de votre projet accède à un type défini dans un autre projet, mais votre projet n'a pas de référence directe au projet de définition.

Le type peut être une classe, une structure, une interface, un module ou une énumération.

Le projet qui définit le type cité produit un assembly contenant le type. Si votre projet ne référence pas directement le projet de définition, le compilateur ne peut pas garantir que l'assembly contenant le type se trouve dans la solution, disponible pour l'accès.

ID d'erreur : BC31532

Pour corriger cette erreur

- Déterminez quel projet définit le type cité et ajoutez une référence à ce projet.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Références de projet
Autres ressources
Références aux espaces de noms et aux composants
Gestion des références

Le projet '<nomprojet>' établit une référence indirecte à l'assembly '<nomassembly>', qui contient '<nomclasse>'

Message d'erreur
Le projet '<nomprojet>' établit une référence indirecte à l'assembly '<nomassembly>', qui contient '<nomclasse>'. Ajoutez une référence à '<nomassembly>' dans votre projet.
Project '<projectname>' makes an indirect reference to assembly '<assemblyname>' which contains '<classname>'. Add a reference to '<assemblyname>' to your project.

Le projet contient des membres qui dépendent d'un assembly non compris dans les références du projet.

ID d'erreur : BC31515

Pour corriger cette erreur

- Ajoutez une référence à l'assembly en cliquant avec le bouton droit sur **Références** dans l'**Explorateur de solutions**, puis en cliquant sur **Ajouter une référence**.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Références de projet

Le projet '<NomProjet>' requiert une référence à la version '<NuméroVersion1>' de l'assembly '<NomAssembly>', mais référence la version '<NuméroVersion2>' de l'assembly '<NomAssembly>' (Avertissement Visual Basic)

Message d'erreur
Le projet '<NomProjet>' requiert une référence à la version '<NuméroVersion1>' de l'assembly '<NomAssembly>', mais référence la version '<NuméroVersion2>' de l'assembly '<NomAssembly>'. Une référence à la version '<NuméroVersion1>' a été émise.
Project '<projectname>' requires a reference to version '<versionnumber1>' of assembly '<assemblyname>', but references version '<versionnumber2>' of assembly '<assemblyname>'. Reference to version '<versionnumber1>' emitted.

Un projet fait une référence indirecte à un assembly qui est défini ailleurs, mais le projet a également une référence directe à une version antérieure de cet assembly.

Pour adapter l'accès aux types et aux éléments de programmation définis dans la version ultérieure mais pas dans la version antérieure, le compilateur utilise la référence indirecte à la version ultérieure lors de la résolution d'accès.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42203

Pour corriger cette erreur

- Supprimez la référence directe à la version antérieure de l'assembly ou modifiez-la pour faire référence à la version ultérieure.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Vue d'ensemble des assemblys
Références de projet
Autres ressources
Références aux espaces de noms et aux composants
Gestion des références

Le projet '<NomProjet>' requiert une référence à la version '<NuméroVersion1>' de l'assembly '<NomAssembly>', mais référence la version '<NuméroVersion2>' de l'assembly '<NomAssembly>' (Erreur Visual Basic)

Project '<projectname>' requires a reference to version '<versionnumber1>' of assembly '<assemblyname>', but references version '<versionnumber2>' of assembly '<assemblyname>' (Visual Basic Error)

Un projet fait une référence indirecte à un assembly qui est défini ailleurs, mais le projet a également une référence directe à une version ultérieure de cet assembly.

Si le compilateur a permis à votre code d'utiliser la référence indirecte, vous ne pouvez peut-être pas accéder aux types d'accès et aux éléments de programmation qui sont définis dans la version ultérieure mais pas dans la version antérieure.

ID d'erreur : BC32209

Pour corriger cette erreur

- Supprimez la référence indirecte à la version antérieure de l'assembly et utilisez la référence directe à la version ultérieure.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Concepts

Vue d'ensemble des assemblys

Références de projet

Autres ressources

Références aux espaces de noms et aux composants

Gestion des références

Le projet '<nomprojet>' ne peut pas faire référence au projet '<nomprojet>', car '<nomprojet>' fait référence directement ou indirectement à '<nomprojet>'

Project '<projectname>' cannot reference project '<projectname>' because '<projectname>' directly or indirectly references '<projectname>'

Un projet fait référence à un second projet. À son tour, le second projet fait référence au projet dans lequel il est référencé.

ID d'erreur : BC30761

Pour corriger cette erreur

- Supprimez une des références.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Le projet '<nomprojet>' ne peut pas générer de référence au fichier '<nomfichier>'

Project '<projectname>' cannot generate a reference to file '<filename>'

Le projet '<*projectname*>' ne peut pas générer de référence au fichier '<*filename*>'. Vous devez peut-être supprimer et rajouter cette référence, puis effectuer une régénération complète de la solution.

Une erreur s'est produite lors de la création d'une référence à un fichier.

ID d'erreur : BC31514

Pour corriger cette erreur

1. Supprimez la référence en cliquant avec le bouton droit sur l'élément des **Références du projet** dans l'**Explorateur de solutions**, puis en cliquant sur **Supprimer**.
2. Restaurez la référence en cliquant avec le bouton droit sur **Références** dans l'**Explorateur de solutions**, puis en cliquant sur **Ajouter une référence**.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio

Concepts

Références de projet

Les accesseurs de propriété ne peuvent pas être déclarés '<MotClé>'

Property accessors cannot be declared '<keyword>'

Une déclaration de procédure **Get** ou **Set** inclut un mot clé qui n'est pas valide sur une procédure de propriété.

L'instruction Get et l'instruction Set (Visual Basic) autorisent uniquement un modificateur d'accès (**Public**, **Protected**, **Friend**, **Protected Friend**, **Private**).

ID d'erreur : BC31099

Pour corriger cette erreur

- Supprimez le mot clé non valide de l'instruction **Get** ou **Set**.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Les accesseurs de propriété ne peuvent pas être déclarés '<ModificateurAccès>' dans une propriété 'NotOverridable'

Property accessors cannot be declared '<accessmodifier>' in a 'NotOverridable' property

Une instruction Get ou Set (Visual Basic) dans une propriété **NotOverridable** contient le mot clé **Private**.

La ligne de raisonnement ci-dessous explique la raison pour laquelle **NotOverridable** et **Private** ne peuvent pas être associés dans une instruction Property :

1. Une propriété ou une procédure qui ne substitue pas de propriété ou de procédure de classe de base a le paramètre par défaut NotOverridable.
2. Toutefois, une propriété ou une procédure dans une classe dérivée qui substitue une propriété ou une procédure de classe de base a le paramètre par défaut Overridable. Pour mettre fin à la hiérarchie de substitution, vous pouvez la déclarer **NotOverridable**. Ce contexte est le seul dans lequel vous pouvez utiliser **NotOverridable**. En d'autres termes, vous pouvez utiliser **NotOverridable** uniquement en association avec Overrides.
3. Si une propriété ou une procédure de classe de base est déclarée Private (Visual Basic), une classe dérivée ne peut pas substituer cette propriété ou procédure car elle ne peut pas y accéder. C'est la raison pour laquelle vous ne pouvez pas utiliser **Private** en association avec **Overridable**.
4. Pour substituer une propriété ou une procédure, la propriété ou la procédure de substitution doit avoir la même signature, mais également le même niveau d'accès. Cela signifie qu'une propriété ou une procédure de substitution ne peut pas spécifier **Private**, car une propriété ou une procédure qui peut être substituée ne peut pas spécifier **Private**.
5. Étant donné que vous pouvez spécifier **NotOverridable** uniquement sur une propriété ou une procédure de substitution, vous ne pouvez pas l'associer à **Private**.

En suivant le même raisonnement, les procédures de propriété (**Get** et **Set**) d'une propriété de substitution ne peuvent pas avoir la valeur **Private**.

ID d'erreur : BC31106

Pour corriger cette erreur

- Supprimez le mot clé **Private** de l'instruction **Get** ou **Set**, ou supprimez les mots clés **Overrides** et **NotOverridable** de l'instruction **Property**.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Différences entre l'occultation et la substitution

Les accesseurs de propriété ne peuvent pas être déclarés '<ModificateurAccès>' dans une propriété 'Default'

Property accessors cannot be declared '<accessmodifier>' in a 'Default' property

Une instructin Get ou instruction Set (Visual Basic) dans une propriété par défaut contient le mot clé **Private**.

Ni une propriété par défaut, ni ses procédures de propriété (**Get** ou **Set**) ne peuvent avoir la valeur **Private**.

ID d'erreur : BC31107

Pour corriger cette erreur

- Supprimez le mot clé **Private** de l'instruction **Get** ou **Set**, ou supprimez **Default** de l'instruction Property.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Comment : déclarer et appeler une propriété par défaut en Visual Basic

Concepts

Procédures Property

Un accès à la propriété doit assigner la propriété ou utiliser sa valeur

Property access must assign to the property or use its value

Vous avez tenté d'accéder à une propriété sans lui assigner de valeur ou sans utiliser cette dernière.

ID d'erreur : BC30545

Pour corriger cette erreur

- Vérifiez la procédure **Property** et spécifiez l'assignation manquante.

Voir aussi

Référence

Opérateurs d'assignation

Concepts

Procédures Property

Impossible de trouver la propriété '<nompropriété>' sans paramètre

Property '<propertyname>' with no parameters cannot be found

L'attribut utilisé possède une propriété qui requiert un ou plusieurs paramètres.

ID d'erreur : BC30658

Pour corriger cette erreur

- Spécifiez les paramètres pour la propriété.

Voir aussi

Autres ressources

Propriétés (Visual Basic)

La propriété '<nompropriété>' doit être déclarée 'Default' car elle se substitue à une propriété par défaut dans la base '<nom>'

Property '<propertyname>' must be declared 'Default' because it overrides a default property on the base '<name>'

Une instruction **Property** de votre code substitue une propriété **Default** dans la classe, structure ou interface de base et doit donc être déclarée **Default**.

ID d'erreur : BC30360

Pour corriger cette erreur

- Déclarez la propriété **Default**.

Voir aussi

Référence

Default (Visual Basic)

La propriété '<nompropriété>' est 'WriteOnly'

Property '<propertyname>' is 'WriteOnly'

Vous avez tenté d'extraire une valeur à partir d'une propriété déclarée comme **WriteOnly**.

ID d'erreur : BC30524

Pour corriger cette erreur

- Supprimez le spécificateur **WriteOnly** de la déclaration de propriété.

Voir aussi

Référence

WriteOnly

La propriété '<nompropriété>' est 'ReadOnly'

Property '<propertyname>' is 'ReadOnly'

Vous avez tenté d'assigner une valeur à une propriété déclarée comme **ReadOnly**.

ID d'erreur : BC30526

Pour corriger cette erreur

- Supprimez le spécificateur **ReadOnly** de la déclaration de propriété.

Voir aussi

Référence

ReadOnly (Visual Basic)

La propriété '<nompropriété>' est d'un type non pris en charge

Property '<propertyname>' is of an unsupported type

Une propriété dans un assembly référencé contient un type qui ne peut pas être représenté.

ID d'erreur : BC30643

Pour corriger cette erreur

- Spécifiez un type qui peut être utilisé avec votre application.

Voir aussi

Concepts

Types de données en Visual Basic

La propriété '<NomPropriété>' déclare implicitement '<NomMembreImplicite>', qui est en conflit avec un membre implicitement déclaré pour le membre '<NomMembre>' dans la classe de base '<NomClasseDeBase>'

Message d'erreur
Le membre '<NomMembre1>' déclare implicitement '<NomMembreImplicite>', qui est en conflit avec un membre implicitement déclaré pour le membre '<NomMembre2>' dans la classe de base '<NomClasseDeBase>'. Par conséquent, le membre ne doit pas être déclaré 'Overloads'.
Member '<membername1>' implicitly declares '<implicitmembername>', which conflicts with a member implicitly declared for member '<membername2>' in the base class '<baseclassname>'. So the member should not be declared 'Overloads'.

Une propriété d'une classe dérivée génère un membre implicite du même nom qu'un membre implicite de la classe de base et spécifie le mot clé Overloads.

La surcharge permet de définir de nombreuses versions d'une propriété ou d'une procédure dans la même classe. Vous ne pouvez pas définir une version supplémentaire d'un membre de classe de base tant que ce membre ne spécifie pas **Overloads**. Dans la mesure où les membres implicites ne spécifient pas **Overloads**, le compilateur suppose que cette propriété Shadows le membre de la classe de base implicite.

Le compilateur Visual Basic crée des membres implicites correspondant à certains éléments de programmation que vous déclarez. Le tableau suivant récapitule ces membres implicit ou *synthetic*.

Élément déclaré	**Membres créés implicitement**
Enumération	Membre `value__`
Événement	Procédure `add_<eventname>` Procédure `remove_<eventname>` Champ `<eventname>Event` Délégué `<eventname>EventHandler`
Propriété	Procédure `get_<propertyname>` Procédure `set_<propertyname>`
Membre de **My.Form**, de **My.WebService** ou d'une classe marquée avec l'attribut MyGroupCollectionAttribute	`m_<variablename>` Variable **Static** Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`

Variable **WithEvents**	Variable `_<variablename>` Propriété `<variablename>` Procédure `get_<variablename>` Procédure `set_<variablename>`

En raison de la possibilité de conflits de noms, vous devez éviter de nommer tout élément de programmation déclaré à l'aide du même formulaire que l'un de ces membres implicites. Vous devez par exemple éviter les noms d'éléments commençant par `get_` ou `set_`.

Ce message est un avertissement par défaut. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40024

Pour corriger cette erreur

- Si vous avez l'intention de masquer ou d'occulter le membre de la classe de base implicite, remplacez le mot clé Overloads par le mot clé Shadows dans la déclaration de la propriété.
- Si vous n'avez pas l'intention d'occulter le membre de la classe de base implicite, modifiez le nom de la propriété pour éviter les conflits de noms répertoriés dans le tableau précédent.

Voir aussi

Concepts

Noms d'éléments déclarés

La propriété ne peut pas être déclarée '<ModificateurPropriété>', car elle contient un accesseur 'Private'

Property cannot be declared '<propertymodifier>' because it contains a 'Private' accessor

Une propriété avec une procédure de propriété **Private** (**Get** ou **Set**) est marquée Overridable.

Si une propriété ou une procédure de classe de base est déclarée Private (Visual Basic), une classe dérivée ne peut pas substituer cette propriété ou procédure car elle ne peut pas y accéder. C'est la raison pour laquelle vous ne pouvez pas utiliser **Private** en association avec **Overridable**. Cela s'applique non seulement à la propriété proprement dite, mais également aux procédures de propriété.

ID d'erreur : BC31108

Pour corriger cette erreur

- Supprimez le mot clé **Overridable** de l'instruction Property, ou supprimez le mot-clé **Private** de l'instruction Get ou Set (Visual Basic).

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Property Get/Let/Set ne sont plus pris en charge ; utilisez la nouvelle syntaxe de déclaration Property.

Property Get/Let/Set are no longer supported; use the new Property declaration syntax

Property Get/Let/Set ne sont plus pris en charge ; utilisez la nouvelle syntaxe de déclaration **Property**.

La syntaxe de déclaration des propriétés a changé. Celles-ci sont définies à présent à l'intérieur des blocs.

ID d'erreur : BC30808

Pour corriger cette erreur

1. Définissez les propriétés dans des blocs de code commençant par le mot clé **Property**. Terminez les propriétés en utilisant la construction **End Property**.
2. Définissez les procédures de propriété **Get** à l'intérieur des blocs de propriété à l'aide du mot clé **Get**. Terminez les procédures de propriété **Get** par la construction **End Get**.
3. Définissez les procédures de propriété **Set** à l'intérieur des blocs de propriété à l'aide du mot clé **Set**. Terminez les procédures de propriété **Set** par la construction **End Set**.
4. Utilisez les procédures de propriété **Set** pour toutes les assignations de propriétés. Les procédures de propriété **Let** ne sont plus nécessaires ni prises en charge.

Voir aussi

Référence

Property, instruction

Autres ressources

Modifications du langage pour les utilisateurs de Visual Basic 6.0

'End Property' manquant dans la propriété

Property missing 'End Property'

Une instruction **End** doit correspondre au bloc qu'elle termine. Une définition **Property** n'a pas été terminée correctement.

ID d'erreur : BC30025

Pour corriger cette erreur

- Ajoutez une instruction **End Property** à la fin de la définition de propriété.

Voir aussi

Référence

End, instruction

Concepts

Procédures Property

La propriété ou le champ '<nom>' n'utilise pas un type d'attribut valide

Property or field '<name>' does not have a valid attribute type

Vous utilisez peut-être un constructeur d'attribut qui n'accepte pas un type d'attribut valide, même s'il est possible que l'attribut possède d'autres constructeurs appropriés.

ID d'erreur : BC30659

Pour corriger cette erreur

- Vérifiez l'utilisation de cet attribut.

Voir aussi

Référence

AttributeUsageAttribute Class

Autres ressources

Attributs en Visual Basic

Les paramètres de propriété ne peuvent pas porter le nom 'Value'

Property parameters cannot have the name 'Value'

Un paramètre dans une instruction **Property** est nommé **Value**. À l'intérieur d'une définition de propriété, ce nom est réservé à la nouvelle valeur assignée à la propriété dans la procédure **Set**.

ID d'erreur : BC32201

Pour corriger cette erreur

- Changez le nom du paramètre dans l'instruction **Property**.

Voir aussi

Référence

Set, instruction (Visual Basic)

Concepts

Procédures Property

Les types protégés ne peuvent être déclarés qu'à l'intérieur d'une classe

Protected types can only be declared inside of a class

Un type à l'intérieur d'un module a été déclaré en tant que **Protected**.

ID d'erreur : BC31047

Pour corriger cette erreur

1. Déclarez le type à l'intérieur d'une classe.
2. Supprimez le modificateur **Protected**.

Voir aussi

Référence

Class, instruction (Visual Basic)
Protected (Visual Basic)

Propriété sans clause 'As' ; type Object pris par défaut

Property without an 'As' clause; type of Object assumed

Une déclaration de propriété ne spécifie pas de clause **As**.

Une clause **As** identifie un type de données à associer à un élément de programmation. Dans l'instruction Property, elle spécifie le type de données de la valeur que la procédure **Get** de la propriété retourne au code appelant. Si vous n'incluez pas une clause **As** dans l'instruction **Property**, le type de données de la propriété a la valeur **Object** par défaut.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42022

Pour corriger cette erreur

- Incluez une clause **As** dans l'instruction **Property** pour spécifier le type de données de la propriété.

Voir aussi

Référence

Property, instruction

Get, instruction

Concepts

Procédures Property

Une propriété sans spécificateur 'ReadOnly' ou 'WriteOnly' doit fournir un 'Get' et un 'Set'

Property without a 'ReadOnly' or 'WriteOnly' specifier must provide both a 'Get' and a 'Set'

Si une propriété n'est pas déclarée comme **ReadOnly** ou **WriteOnly**, elle doit fournir des procédures pour la lecture et l'écriture de sa valeur.

ID d'erreur : BC30124

Pour corriger cette erreur

1. Assurez-vous que d'inclure une procédure **Get** et une procédure **Set** entre l'instruction **Property** et l'instruction **End Property**.
2. Vérifiez que les autres procédures à l'intérieur de la déclaration **Property** sont correctement terminées.

Voir aussi

Référence

Property, instruction
Get, instruction
Set, instruction (Visual Basic)

La déclaration 'RaiseEvent' doit se terminer par un 'End RaiseEvent' correspondant

'RaiseEvent' declaration must end with a matching 'End RaiseEvent'

Une déclaration **RaiseEvent** doit se terminer par une instruction **End RaiseEvent**

ID d'erreur : BC31117

Pour corriger cette erreur

- Vérifiez que la déclaration **RaiseEvent** se termine par une instruction **End RaiseEvent**

Voir aussi

Référence

RaiseEvent, instruction
Event, instruction

La définition 'RaiseEvent' est manquante pour l'événement '<nomévénement>'

'RaiseEvent' definition missing for event '<eventname>'

Si un événement est déclaré en tant que **Custom**, il doit fournir une procédure pour déclencher l'événement.

ID d'erreur : BC31132

Pour corriger cette erreur

1. Incluez une déclaration **RaiseEvent** entre l'instruction **Custom Event** et l'instruction **End Event**.
2. Vérifiez que les autres procédures à l'intérieur de la déclaration event sont correctement terminées.

Voir aussi

Référence

RaiseEvent, instruction
Event, instruction

'RaiseEvent' est déjà déclaré

'RaiseEvent' is already declared

Plusieurs déclarations **RaiseEvent** sont apparues dans une déclaration event personnalisée. Une déclaration **RaiseEvent** déclare une procédure utilisée pour déclencher l'événement.

ID d'erreur : BC31129

Pour corriger cette erreur

- Supprimez l'instruction **RaiseEvent** redondante.

Voir aussi

Référence

RaiseEvent, instruction
Event, instruction

La méthode 'RaiseEvent' doit avoir la même signature que le type délégué de l'événement conteneur '<signature>'

'RaiseEvent' method must have the same signature as the containing event's delegate type '<signature>'

Une déclaration **Custom Event** doit posséder une déclaration **RaiseEvent** qui a la même signature que le type délégué spécifié par la clause **As** de l'événement personnalisé.

Pour que les signatures correspondent, la déclaration **RaiseEvent** et le délégué doivent avoir le même nombre de paramètres, et les types de paramètres doivent correspondre.

ID d'erreur : BC31137

Pour corriger cette erreur

- Modifiez les paramètres de la déclaration **RaiseEvent** pour les faire correspondre aux paramètres du type délégué.

Exemple

Cet exemple illustre un événement personnalisé avec les types de paramètres corrects pour la déclaration **RaiseEvent**.

VB

```
Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)
Custom Event Test As TestDelegate
    AddHandler(ByVal value As TestDelegate)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As TestDelegate)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal i As Integer)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

Voir aussi

Référence

Event, instruction
RaiseEvent
Delegate, instruction

Autres ressources

Événements en Visual Basic

Les instructions 'RaiseEvent' ne sont pas valides dans la fenêtre Exécution

'RaiseEvent' statements are not valid in the Immediate window

Vous ne pouvez pas utiliser les instructions **RaiseEvent** dans la fenêtre **Exécution**.

ID d'erreur : BC30739

Pour corriger cette erreur

- Testez les instructions **RaiseEvent** du code dans l'**éditeur de code**.

Voir aussi

Référence

RaiseEvent, instruction

Autres ressources

Débogage dans Visual Studio

La plage spécifiée pour l'instruction 'Case' n'est pas valide

Range specified for 'Case' statement is not valid

Une plage non valide a été spécifiée pour une instruction **Case**.

Si vous comparez la même expression à plusieurs valeurs différentes, vous pouvez utiliser les instructions **Select...Case** plutôt que les instructions **If...Then...Else**. Tandis que les instructions **If** et **ElseIf** permettent d'évaluer une expression différente dans chaque instruction, l'instruction **Select** évalue une expression unique une seule fois et l'utilise pour chaque comparaison. Chaque instruction **Case** peut contenir plusieurs valeurs, une plage de valeurs ou une combinaison de valeurs et d'opérateurs de comparaison.

ID d'erreur : BC40052

Pour corriger cette erreur

- Modifiez la plage pour inclure toutes les valeurs, ou utilisez une instruction **Case Else** pour intercepter une valeur non définie.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

Concepts

Structures de décision

Conversions étendues et restrictives

Impossible de combiner 'ReadOnly' et 'WriteOnly'

'ReadOnly' and 'WriteOnly' cannot be combined

Les modificateurs **ReadOnly** et **WriteOnly** s'excluent mutuellement.

ID d'erreur : BC30635

Pour corriger cette erreur

- Spécifiez **ReadOnly** ou **WriteOnly**, mais pas les deux.

Voir aussi

Référence

ReadOnly (Visual Basic)
WriteOnly

La propriété d'attribut 'ReadOnly' '<champpropriété>' ne peut pas être la cible d'une assignation

'ReadOnly' attribute property '<propertyfield>' cannot be the target of an assignment

Une tentative d'assignation d'une valeur à une propriété **ReadOnly** dans un attribut a été effectuée.

ID d'erreur : BC31501

Pour corriger cette erreur

1. Supprimez l'instruction d'assignation de propriété.
2. Si vous utilisez des propriétés que vous avez développées, supprimez les modificateurs **ReadOnly** ou **Shared** de la propriété d'attribut.

Voir aussi

Référence

Shared (Visual Basic)

Autres ressources

Attributs en Visual Basic

Les propriétés 'ReadOnly' ne peuvent pas avoir un modificateur d'accès pour 'Get'

'ReadOnly' properties cannot have an access modifier on 'Get'

Une déclaration de propriété **ReadOnly** spécifie des niveaux d'accès dans l'instruction Property et l'instruction Get .

Vous pouvez toujours spécifier un niveau d'accès pour la propriété. De plus, vous pouvez spécifier un niveau d'accès différent pour l'une de ses procédures de propriété (**Get** ou **Set**), à condition qu'il soit plus restrictif que le niveau d'accès de la propriété. Vous ne pouvez pas spécifier de niveaux d'accès pour les deux procédures de propriété.

ID d'erreur : BC31105

Pour corriger cette erreur

- Supprimez le modificateur d'accès de l'instruction **Get**. Il représente la propriété **ReadOnly** entière, et vous ne pouvez pas définir deux niveaux d'accès pour la propriété.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

La propriété 'ReadOnly' '<nompropriété>' ne peut pas être la cible d'une assignation

'ReadOnly' property '<propertyname>' cannot be the target of an assignment

Une propriété **ReadOnly** apparaît dans un contexte qui lui assigne une valeur. Seuls les variables, propriétés et éléments de tableaux accessibles en écriture peuvent avoir des valeurs qui leur sont assignées pendant l'exécution.

ID d'erreur : BC30098

Pour corriger cette erreur

- Supprimez le mot clé **ReadOnly** de l'instruction **Property** déclarant la variable ou supprimez l'instruction qui lui assigne une valeur.

Voir aussi

Référence

ReadOnly (Visual Basic)
Property, instruction

Une propriété 'ReadOnly' doit fournir un 'Get'

'ReadOnly' property must provide a 'Get'

Si une propriété est déclarée comme **ReadOnly**, elle doit fournir une procédure pour la lecture de sa valeur.

ID d'erreur : BC30126

Pour corriger cette erreur

1. Veillez à inclure une procédure **Get** entre l'instruction **Property** et l'instruction **End Property**.
2. Vérifiez que les autres procédures à l'intérieur de la déclaration **Property** sont correctement terminées.

Voir aussi

Référence

Property, instruction
Get, instruction

Une variable 'ReadOnly' ne peut pas être la cible d'une assignation

'ReadOnly' variable cannot be the target of an assignment

Une propriété **ReadOnly** a été trouvée dans un contexte qui lui assigne une valeur. Seuls les variables, propriétés et éléments de tableaux accessibles en écriture peuvent avoir des valeurs qui leur sont assignées pendant l'exécution.

ID d'erreur : BC30064

Pour corriger cette erreur

- Supprimez le mot clé **ReadOnly** de l'instruction **Dim** déclarant la variable ou supprimez l'instruction qui lui assigne une valeur.

Voir aussi

Référence

ReadOnly (Visual Basic)
Dim, instruction (Visual Basic)

'Redim' ne peut pas changer le nombre de dimensions d'un tableau

'ReDim' cannot change the number of dimensions of an array

Vous avez tenté d'utiliser **ReDim** pour modifier le rang (nombre de dimensions) d'un tableau. L'instruction **ReDim** peut être utilisée pour changer la taille d'une ou plusieurs dimensions d'un tableau qui a déjà été déclaré formellement, mais elle ne peut pas changer le rang d'un tableau.

ID d'erreur : BC30415

Pour corriger cette erreur

- Vérifiez que vous souhaitez obtenir le rang au lieu des tailles du tableau et, si possible, utilisez **Dim** pour déclarer un nouveau tableau avec le rang souhaité.

Voir aussi

Référence

ReDim, instruction (Visual Basic)

Dim, instruction (Visual Basic)

Concepts

Vue d'ensemble des tableaux dans Visual Basic

L'objet référencé a la valeur 'Nothing'

Referenced object has a value of 'Nothing'

L'objet utilisé a la valeur **Nothing**, mais une valeur utilisable est attendue. **Nothing** est une valeur qui indique qu'un objet n'a pas de valeur, soit parce que vous ne lui en avez assigné aucune, soit parce que vous lui avez assigné la valeur **Nothing**.

ID d'erreur : BC30760

Pour corriger cette erreur

1. Vérifiez que l'objet a été déclaré dans la portée de la procédure à l'endroit où l'erreur s'est produite.
2. Vérifiez que la valeur de l'objet est définie.

Remarque
La valeur **Nothing** est différente de la valeur zéro ou d'une chaîne vide. Vous pouvez utiliser **IsNothing** afin de vérifier si un objet contient la valeur **Nothing**.

Voir aussi

Référence

Nothing (Visual Basic)
IsNothing, fonction

L'objet référencé '<nomobjet>' a la valeur 'Nothing'

Referenced object '<objectname>' has a value of 'Nothing'

L'objet référencé n'a pas été défini ou a pour valeur **Nothing**.

ID d'erreur : BC30705

Pour corriger cette erreur

- Vérifiez que le nom de l'objet est correctement orthographié et que l'objet a été défini.

Voir aussi

Référence

Nothing (Visual Basic)

Autres ressources

Débogage dans Visual Studio

L'élément de tableau référencé a la valeur 'Nothing'

Referenced array element has a value of 'Nothing'

L'élément de tableau référencé n'a pas été défini ou avait pour valeur **Nothing**.

ID d'erreur : BC30744

Pour corriger cette erreur

- Vérifiez que les valeurs du tableau sont correctement définies.

Voir aussi

Référence

Nothing (Visual Basic)

Le '<nomobjet>' référencé a la valeur 'Nothing'

Referenced '<membername>' has a value of 'Nothing'

Soit vous n'avez assigné aucune valeur à l'élément référencé, soit vous lui avez assigné la valeur **Nothing**.

ID d'erreur : BC30759

Pour corriger cette erreur

- Vérifiez que le nom de l'élément est correctement orthographié et défini.

Voir aussi

Référence

Nothing (Visual Basic)

Toute référence à un objet en construction n'est pas valide lors de l'appel d'un autre constructeur

Reference to object under construction is not valid when calling another constructor

Une référence à un membre objet a été faite avant que le constructeur de l'objet ait terminé la création de l'objet.

ID d'erreur : BC31095

Pour corriger cette erreur

- N'utilisez pas **MyBase**, **MyClass** ou **Me** lorsque vous appelez un constructeur à partir d'un autre constructeur.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

Concepts

Durée de vie d'un objet : création et destruction des objets

Une référence à un membre non partagé requiert une référence d'objet

Reference to a non-shared member requires an object reference

Vous avez fait référence à un membre non partagé dans votre code et vous n'avez pas réussi à fournir une référence d'objet. Vous ne pouvez pas utiliser le nom de la classe pour qualifier un membre qui n'est pas partagé. L'instance doit d'abord être déclarée comme une variable objet et ensuite référencée par le nom de la variable.

ID d'erreur : BC30469

Pour corriger cette erreur

1. Déclarez l'instance comme une variable objet.
2. Faites référence à l'instance par le nom de la variable.

Voir aussi

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

Une référence à l'assembly '<IdentitéAssembly>' contenant le type '<NomType>' est requise, mais une référence adéquate n'a pas été trouvée en raison de l'ambiguïté entre les projets '<NomProjet1>' et '<NomProjet2>'

Reference required to assembly '<assemblyidentity>' containing type '<typename>', but a suitable reference could not be found due to ambiguity between projects '<projectname1>' and '<projectname2>'

Une expression utilise un type, tel qu'une classe, une structure, une interface, une énumération ou un délégué qui est défini à l'extérieur de votre projet. Toutefois, vous avez des références de projet à plusieurs assemblys définissant ce type.

Les projets cités produisent des assemblys avec le même nom. À cause de ceci, le compilateur ne peut pas déterminer quel assembly utiliser pour le type auquel vous accédez.

Pour accéder à un type défini dans un autre assembly, le compilateur Visual Basic doit avoir une référence à cet assembly. Il doit s'agir d'une référence unique et non équivoque qui ne provoque pas de références circulaires parmi des projets.

ID d'erreur : BC30969

Pour corriger cette erreur

1. Déterminez quel projet produit le meilleur assembly pour votre projet à référencer. Pour prendre cette décision, vous pouvez utiliser des critères, tels que la facilité d'accès au fichier et la fréquence des mises à jour.
2. Dans vos propriétés de projet, ajoutez une référence au fichier contenant l'assembly qui définit le type que vous utilisez.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio
Comment : modifier des propriétés de projet et des paramètres de configuration
Dépannage de références rompues

Concepts

Références de projet
Résolution d'une référence lorsque plusieurs variables ont le même nom

Autres ressources

Références aux espaces de noms et aux composants

Une référence à l'assembly '<IdentitéAssembly>' contenant le type '<NomType>' est requise, mais une référence adéquate n'a pas été trouvée en raison de références circulaires possibles : <ListeDépendanceRéférence>

Reference required to assembly '<assemblyidentity>' containing type '<typename>', but a suitable reference could not be found due to possible circular references: <referencedependencylist>

Une expression utilise un type, tel qu'une classe, une structure, une interface, une énumération ou un délégué qui est défini à l'extérieur de votre projet. Toutefois, votre référence de projet à cet assembly fait partie d'un jeu de références circulaires.

Lorsque plusieurs projets ont des références entre eux, il est possible que les références soient *circulaires*. Par exemple, deux projets peuvent avoir des références l'un à l'autre. De manière générale, une chaîne de références d'un projet au suivant peut revenir au projet initial. Dans ces cas, il n'y a pas de dernier projet à la fin de la chaîne avec lequel résoudre la référence.

Pour accéder à un type défini dans un autre assembly, le compilateur Visual Basic doit avoir une référence à cet assembly. Il doit s'agir d'une référence unique et non équivoque qui ne provoque pas de références circulaires parmi des projets.

ID d'erreur : BC30962

Pour corriger cette erreur

- Dans vos propriétés de projet, ajoutez une référence directe au projet produisant l'assembly qui définit le type que vous utilisez.

Voir aussi

Tâches

Comment : ajouter ou supprimer des références en Visual Studio
Comment : modifier des propriétés de projet et des paramètres de configuration
Dépannage de références rompues

Concepts

Références de projet

Autres ressources

Références aux espaces de noms et aux composants

Une référence au module '<nommodule>' contenant le type '<nommembre>' est requise

Message d'erreur
Une référence au module '<nommodule>' contenant le type '<nommembre>' est requise. Ajoutez-en une à votre projet.
Reference required to module '<modulename>' containing the type '<membername>'. Add one to your project.

Le projet contient un type qui dépend d'un autre type provenant d'un assembly non inclus dans les références du projet.

ID d'erreur : BC30653

Pour corriger cette erreur

- Ajoutez une référence de projet pour les types dépendants du projet.

Voir aussi

Concepts
Références de projet

Une référence au module '<nommodule>' contenant l'interface implémentée '<nominterface>' est requise

Message d'erreur
Une référence au module '<nommodule>' contenant l'interface implémentée '<nominterface>' est requise. Ajoutez-en une à votre projet.
Reference required to module '<modulename>' containing the implemented interface '<interfacename>'. Add one to your project.

L'interface est définie dans un module non directement référencé dans votre projet. Le compilateur Visual Basic requiert une référence afin d'éviter toute ambiguïté au cas où l'interface serait définie dans plusieurs modules.

ID d'erreur : BC30010

Pour corriger cette erreur

- Incluez le nom du module non référencé dans vos références de projet.

Voir aussi

Tâches
Dépannage de références rompues
Autres ressources
Références aux espaces de noms et aux composants

Une référence au module '<nommodule>' contenant la définition pour l'événement '<nomévénement>' est requise

Reference required to module '<modulename>' containing the definition for event '<eventname>'

Une référence au module '<*modulename*>' contenant la définition pour l'événement '<*eventname*>' est requise. Ajoutez-en une à votre projet.

L'événement est défini dans un module non directement référencé dans votre projet. Le compilateur Visual Basic requiert une référence afin d'éviter toute ambiguïté au cas où l'événement serait défini dans plusieurs modules.

ID d'erreur : BC30006

Pour corriger cette erreur

- Incluez le nom du module non référencé dans vos références de projet.

Voir aussi

Tâches

Dépannage de références rompues

Autres ressources

Références aux espaces de noms et aux composants

Une référence au module '<nommodule>' contenant la classe de base '<nomclasse>' est requise

Message d'erreur
Une référence au module '<nommodule>' contenant la classe de base '<nomclasse>' est requise. Ajoutez-en une à votre projet.
Reference required to module '<modulename>' containing the base class '<classname>'. Add one to your project.

La classe est définie dans un module non directement référencé dans votre projet. Le compilateur Visual Basic requiert une référence afin d'éviter toute ambiguïté au cas où la classe serait définie dans plusieurs modules.

ID d'erreur : BC30008

Pour corriger cette erreur

- Incluez le nom du module non référencé dans vos références de projet.

Voir aussi

Tâches
Dépannage de références rompues
Autres ressources
Références aux espaces de noms et aux composants

Une référence à l'assembly '<nomassembly>' contenant le type '<nommembre>' est requise

Message d'erreur
Une référence à l'assembly '<nomassembly>' contenant le type '<nommembre>' est requise. Ajoutez-en une à votre projet.
Reference required to assembly '<assemblyname>' containing the type '<membername>'. Add one to your project.

Le projet contient un type qui dépend d'un autre type provenant d'un assembly non inclus dans les références du projet.

ID d'erreur : BC30652

Pour corriger cette erreur

- Ajoutez une référence de projet pour les types dépendants du projet.

Voir aussi

Concepts
Références de projet

Une référence à l'assembly '<nomassembly>' contenant l'interface implémentée '<nominterface>' est requise

Message d'erreur
Une référence à l'assembly '<nomassembly>' contenant l'interface implémentée '<nominterface>' est requise. Ajoutez-en une à votre projet.
Reference required to assembly '<assemblyname>' containing the implemented interface '<interfacename>'. Add one to your project.

L'interface est définie dans une DLL ou un assembly non directement référencé dans votre projet. Le compilateur Visual Basic requiert une référence afin d'éviter toute ambiguïté au cas où l'interface serait définie dans plusieurs DLL ou assemblys.

ID d'erreur : BC30009

Pour corriger cette erreur

- Incluez le nom de la DLL ou de l'assembly non référencé dans vos références de projet.

Voir aussi

Tâches
Dépannage de références rompues
Autres ressources
Références aux espaces de noms et aux composants

Une référence à l'assembly '<nomassembly>' contenant la définition de l'événement '<nomévénement>' est requise

Reference required to assembly '<assemblyname>' containing the definition for event '<eventname>'

Une référence à l'assembly '<*assemblyname*>' contenant la définition de l'événement '<*eventname*>' est requise. Ajoutez une référence à votre projet.

L'événement est défini dans une bibliothèque de liens dynamiques (DLL, Dynamic Link Library) ou un assembly non directement référencé dans votre projet. Le compilateur Visual Basic requiert une référence afin d'éviter toute ambiguïté au cas où l'événement serait défini dans plusieurs DLL ou assemblys.

ID d'erreur : BC30005

Pour corriger cette erreur

- Incluez le nom de la DLL ou de l'assembly non référencé dans vos références de projet.

Voir aussi

Tâches

Dépannage de références rompues

Autres ressources

Références aux espaces de noms et aux composants

Une référence à l'assembly '<nomassembly>' contenant la classe de base '<nomclasse>' est requise

Message d'erreur
Une référence à l'assembly '<nomassembly>' contenant la classe de base '<nomclasse>' est requise. Ajoutez-en une à votre projet.
Reference required to assembly '<assemblyname>' containing the base class '<classname>'. Add one to your project.

La classe est définie dans une DLL ou un assembly non directement référencé dans votre projet. Le compilateur Visual Basic requiert une référence afin d'éviter toute ambiguïté au cas où la classe serait définie dans plusieurs DLL ou assemblys.

ID d'erreur : BC30007

Pour corriger cette erreur

- Incluez le nom de la DLL ou de l'assembly non référencé dans vos références de projet.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Dépannage de références rompues
Concepts
Références de projet
Autres ressources
Références aux espaces de noms et aux composants

Les instructions 'ReDim' requièrent une liste entre parenthèses des nouvelles limites de chaque dimension du tableau

'ReDim' statements require a parenthesized list of the new bounds of each dimension of the array

Vous devez spécifier la nouvelle taille d'un tableau en tant que partie d'une instruction **ReDim**.

ID d'erreur : BC30670

Pour corriger cette erreur

- Indiquez la taille de chaque rang du tableau. Par exemple :

```
ReDim arr(5, 6)
```

Voir aussi

Référence

ReDim, instruction (Visual Basic)

Les instructions 'ReDim' ne peuvent plus être utilisées pour déclarer des variables tableau

'ReDim' statements can no longer be used to declare array variables

Les instructions **ReDim** ne peuvent être utilisées que pour changer la taille d'un tableau existant.

ID d'erreur : BC30811

Pour corriger cette erreur

- Spécifiez la taille des tableaux lors de leur déclaration. Par exemple :

```
Dim X(20) As Integer
```

Voir aussi

Référence

Liste des tableaux

ReDim, instruction (Visual Basic)

Concepts

Instruction ReDim pour les utilisateurs de Visual Basic 6.0

'RemoveHandler' est déjà déclaré

'RemoveHandler' is already declared

Plusieurs déclarations **RemoveHandler** sont apparues dans une déclaration event personnalisée. Une déclaration **RemoveHandler** déclare la procédure utilisée pour supprimer un gestionnaire d'événements.

ID d'erreur : BC31128

Pour corriger cette erreur

- Supprimez l'instruction **RemoveHandler** redondante.

Voir aussi

Référence

RemoveHandler, instruction
Event, instruction

La définition 'RemoveHandler' est manquante pour l'événement '<nomévénement>'

'RemoveHandler' definition missing for event '<eventname>'

Si un événement est déclaré en tant que **Custom**, il doit fournir une procédure pour supprimer un gestionnaire d'événements.

ID d'erreur : BC31131

Pour corriger cette erreur

1. Incluez une déclaration **RemoveHandler** entre l'instruction **Custom Event** et l'instruction **End Event**.
2. Vérifiez que les autres procédures à l'intérieur de la déclaration event sont correctement terminées.

Voir aussi

Référence

RemoveHandler, instruction
Event, instruction

La déclaration 'RemoveHandler' doit se terminer par un 'End RemoveHandler' correspondant

'RemoveHandler' declaration must end with a matching 'End RemoveHandler'

Une déclaration **RemoveHandler** doit se terminer par une instruction **End RemoveHandler**

ID d'erreur : BC31116

Pour corriger cette erreur

- Vérifiez que la déclaration **RemoveHandler** se termine par une instruction **End RemoveHandler**

Voir aussi

Référence

RemoveHandler, instruction
Event, instruction

Opérateur relationnel attendu

Relational operator expected

Une instruction **Case** contient une clause **Is**, mais pas d'opérateur de comparaison, tel que **=** ou **>**. Si une instruction **Case** ne comporte pas d'opérateur, **=** est supposé et **Is** n'est pas utilisé.

ID d'erreur : BC30239

Pour corriger cette erreur

- Supprimez le mot clé **Is** ou faites-le suivre d'un opérateur de comparaison.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

Opérateurs de comparaison (Visual Basic)

Concepts

Opérateurs de comparaison en Visual Basic

Les instructions 'Resume' ne sont pas valides dans la fenêtre Exécution

'Resume' statements are not valid in the Immediate window

Les instructions Resume ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30714

Pour corriger cette erreur

- Supprimez les instructions **Resume** du code de débogage.

Voir aussi

Référence

Resume

Autres ressources

Débogage dans Visual Studio

'Resume' ou 'GoTo' attendu

'Resume' or 'GoTo' expected

Une instruction **On Error** ne contient pas de clause **Resume** ou **GoTo**. Elle doit utiliser l'une de ces deux clauses.

ID d'erreur : BC32019

Pour corriger cette erreur

- Ajoutez une clause **Resume** ou une clause **GoTo** à l'instruction **On Error**.

Voir aussi

Référence

On Error, instruction (Visual Basic)
Resume, instruction
GoTo, instruction

Fichier réponse <nomfichier> inclus plusieurs fois

Response file <filename> included multiple times

Un fichier réponse a été spécifié plusieurs fois. Les fichiers réponse contiennent des options du compilateur et des fichiers de code source à compiler.

ID d'erreur : BC2003

Pour corriger cette erreur

- Supprimez les spécificateurs de fichiers réponse redondants.

Voir aussi

Référence

Options du compilateur Visual Basic par catégorie

Le nom de ressource '<nomressource>' ne peut pas être utilisé plusieurs fois

Resource name '<resourcename>' cannot be used more than once

Vous avez utilisé plusieurs fois le nom d'une ressource spécifiée dans une bibliothèque de liens dynamiques ou dans un fichier exécutable.

ID d'erreur : BC31502

Pour corriger cette erreur

- Nommez les ressources de manière unique.

Voir aussi

Référence

/linkresource (Visual Basic)
/resource (Visual Basic)

Le type de retour de la fonction '<NomProcédure>' n'est pas conforme CLS

Return type of function '<procedurename>' is not CLS-compliant

Une procédure **Function** est marquée comme `<CLSCompliant(True)>`, mais retourne un type qui est marqué comme `<CLSCompliant(False)>`, qui n'est pas marqué ou qui ne qualifie pas car son type n'est pas conforme.

Pour qu'une procédure soit conforme au Spécification CLS (Common Language Specification) (CLS), elle doit utiliser uniquement des types conformes CLS. Cela s'applique aux types des paramètres, au type de retour et aux types de toutes ses variables locales.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40027

Pour corriger cette erreur

- Si la procédure **Function** doit retourner ce type particulier, supprimez **CLSCompliantAttribute**. La procédure ne peut pas être conforme CLS.
- Si la procédure **Function** doit être conforme CLS, remplacez le type de retour par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.
- Si vous utilisez des objets Automation ou COM, n'oubliez pas que certains types ont des largeurs de données différentes de celles dans le .NET Framework. Par exemple, **int** correspond souvent à 16 bits dans d'autres environnements. Si vous retournez un entier de 16 bits à un tel composant, déclarez-le comme type de données **Short** et non comme **Integer** dans votre code Visual Basic managé.

Voir aussi

Concepts

Écriture d'un code conforme CLS

Les instructions 'Return' ne sont pas valides dans la fenêtre Exécution

'Return' statements are not valid in the Immediate window

L'instruction **Return** effectue un branchement et n'est pas autorisée dans un contexte de débogage.

ID d'erreur : BC30147

Pour corriger cette erreur

- N'émettez pas d'instruction **Return** dans la fenêtre **Exécution**.

Voir aussi

Référence

Fenêtre Exécution

L'instruction 'Return' dans une méthode 'AddHandler', 'RemoveHandler' ou 'RaiseEvent' ne peut pas retourner une valeur

'Return' statement in an 'AddHandler', 'RemoveHandler', or 'RaiseEvent' method cannot return a value

Les méthodes **AddHandler**, **RemoveHandler** et **RaiseEvent** dans une déclaration **Custom Event** peut contenir des instructions **Return** pour quitter les méthodes. Toutefois, l'instruction **Return** ne peut pas spécifier de valeur de retour parce que les méthodes ne peuvent pas retourner de valeurs.

ID d'erreur : BC30940

Pour corriger cette erreur

- Supprimez l'expression qui suit l'instruction **Return**.

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler
RaiseEvent
Return, instruction (Visual Basic)

Autres ressources

Événements en Visual Basic

L'instruction 'Return' dans un Sub ou un Set ne peut pas retourner de valeur

'Return' statement in a Sub or a Set cannot return a value

Les procédures**Sub** et les procédures de propriété **Set** ne peuvent pas retourner de valeur.

ID d'erreur : BC30647

Pour corriger cette erreur

- Remplacez la procédure en cours par une fonction, ou par une procédure de propriété **Get** si la procédure en cours fait partie d'une propriété.
- Vous pouvez en effet retourner des valeurs à partir de procédures **Sub** en modifiant la valeur des paramètres passés par référence à l'aide du mot clé **ByRef**.

Voir aussi

Référence

Return, instruction (Visual Basic)

Concepts

Sub, procédures

Function, procédures

Procédures Property

L'instruction 'Return' dans un Function, Get ou Operator doit retourner une valeur

'Return' statement in a Function, Get, or Operator must return a value

Les instructions **Return** doivent être utilisées pour retourner une valeur à une procédure appelante. Vous ne pouvez pas utiliser d'instructions **Return** pour contrôler le déroulement du programme.

ID d'erreur : BC30654

Pour corriger cette erreur

1. Spécifiez une valeur pouvant être retournée par la fonction ou la procédure.
2. Utilisez l'instruction **End** pour que le programme quitte la procédure en cours.

Voir aussi

Référence

Return, instruction (Visual Basic)
End (Visual Basic)

Les types de retour et de paramètres de '<opérateur>' doivent être '<NomType>' pour être utilisés dans une instruction 'For'

Return and parameter types of '<operator>' must be '<typename>' to be used in a 'For' statement

Une boucle **For** spécifie une variable de compteur d'un type qui ne définit pas l'opérateur **+** ou **-** avec les paramètres et la valeur de retour de son propre type.

La variable de compteur doit être d'un type qui prend en charge les opérateurs d'addition (**+**) et de soustraction (**-**) qui fonctionnent entièrement sur leur type conteneur. Cela signifie que les opérandes et la valeur de retour doivent être du type de la variable de compteur.

Si vous utilisez un type de données numériques pour la variable de compteur, les opérateurs **+** et **-** sont pris en charge sur le type conteneur. Si vous utilisez une classe ou une structure définie par l'utilisateur, vous devez définir les deux opérateurs avec des opérandes et une valeur de retour du type de votre classe ou structure.

ID d'erreur : BC33039

Pour corriger cette erreur

1. Assurez-vous que l'orthographe du type de données de la variable de compteur est correcte.
2. Si vous utilisez une classe ou une structure définie par l'utilisateur pour la variable de compteur, définissez les opérateurs **+** et **-** qui fonctionnent entièrement sur cette classe ou structure.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

For...Next, instruction (Visual Basic)
Operator, instruction

Concepts

Procédures d'opérateur

Les types de paramètres et de retour de '<OpérateurLogique>' doivent être de type '<NomType>' pour être utilisés dans une expression '<OpérateurCourtCircuit>'

Return and parameter types of '<logicaloperator>' must be '<typename>' to be used in a '<shortcircuitoperator>' expression

Un opérateur **And** ou **Or** est déclaré avec des paramètres inadéquats ou un type de retour destiné à une utilisation dans un opérateur AndAlso ou OrElse.

Étant donné que vous ne définissez pas directement un opérateur de court-circuit (**AndAlso** ou **OrElse**), vous devez définir les opérateurs logiques et déterminants correspondants. Le tableau suivant présente les opérateurs requis.

Opérateur de court-circuit	Opérateur logique	Opérateur déterminant
AndAlso	And, opérateur (Visual Basic)	IsFalse, opérateur
OrElse	Or, opérateur (Visual Basic)	IsTrue, opérateur

Visual Basic utilise ces opérateurs logiques et déterminants pour construire la logique « court-circuitante » pour **AndAlso** ou **OrElse**. Pour que cela fonctionne correctement, les opérandes et la valeur de retour de votre définition **And** ou **Or** doivent être du type conteneur, c'est-à-dire du type de la classe ou la structure dans laquelle vous définissez **And** ou **Or**.

ID d'erreur : BC33034

Pour corriger cette erreur

- Remplacez le type des deux opérandes et de la valeur de retour par le type de la classe ou la structure dans laquelle vous définissez cet opérateur.

  - ou -

- N'utilisez pas l'opérateur de court-circuit correspondant (**AndAlso** ou **OrElse**) avec les opérandes du type de la classe ou la structure dans laquelle vous définissez cet opérateur **And** ou **Or**.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Opérateurs de bits et opérateurs logiques en Visual Basic

Exception runtime levée : <exception> - <exception>

Run-time exception thrown: <exception> - <exception>

L'une des expressions de la fenêtre de débogage a déclenché une erreur au moment de l'exécution.

ID d'erreur : BC30703

Pour corriger cette erreur

- Vérifiez le code approprié.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Exception runtime levée

Run-time exception thrown

L'une des expressions de la fenêtre de débogage a déclenché une erreur au moment de l'exécution.

ID d'erreur : BC30704

Pour corriger cette erreur

- Vérifiez le code.

Voir aussi

Autres ressources

Débogage dans Visual Studio

L'espace de noms racine <NomEspaceDeNoms> n'est pas conforme CLS

Root namespace <namespacename> is not CLS-compliant

Un assembly est marqué comme `<CLSCompliant(True)>`, mais le nom d'espace de noms racine commence par un trait de soulignement (_).

Un élément de programmation peut contenir un ou plusieurs traits de soulignement, mais pour être conforme à la Spécification CLS (Common Language Specification) (CLS), il ne doit pas commencer par un trait de soulignement. Consultez Noms d'éléments déclarés.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40038

Pour corriger cette erreur

- S'il est nécessaire d'avoir la conformité CLS, modifiez le nom de l'espace de noms racine de sorte qu'il ne commence pas par un trait de soulignement.
- Si le nom de l'espace de noms racine doit rester inchangé, supprimez l'**CLSCompliantAttribute** de l'assembly ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Tâches

Comment : modifier l'espace de noms pour une application

Référence

Namespace, instruction

/rootnamespace

Concepts

Espaces de noms dans Visual Basic

Noms d'éléments déclarés

Conventions d'affectation de noms Visual Basic

Écriture d'un code conforme CLS

La méthode 'Set' ne peut pas comporter plusieurs paramètres

'Set' method cannot have more than one parameter

Plusieurs paramètres sont définis pour les procédures de propriétés **Set**.

ID d'erreur : BC31063

Pour corriger cette erreur

- Supprimez tout paramètre supplémentaire des procédures de propriété **Set**.

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Concepts

Procédures Property

Propriétés et procédures Property

'Set' est déjà déclaré

'Set' is already declared

Une instruction **Set** apparaît plusieurs fois dans un bloc de code. L'instruction **Set** déclare une procédure de propriété **Set** utilisée pour assigner une valeur à une propriété.

ID d'erreur : BC30444

Pour corriger cette erreur

- Supprimez l'instruction **Set** redondante.

Voir aussi

Référence

Set, instruction (Visual Basic)

L'accesseur 'Set' de la propriété '<NomPropriété>' n'est pas accessible

'Set' accessor of property '<propertyname>' is not accessible

Une instruction essaie d'enregistrer la valeur d'une propriété alors qu'elle n'a pas accès à la procédure **Set** de la propriété.

Si l'instruction Set (Visual Basic) est marquée avec un niveau d'accès plus restrictif que son instruction Property, une tentative de définition de la valeur de la propriété peut échouer dans les cas suivants :

- L'instruction **Set** est marquée Private (Visual Basic) et le code appelant est situé à l'extérieur de la classe ou la structure dans laquelle la propriété est définie.
- L'instruction **Set** est marquée Protected (Visual Basic) et le code appelant ne figure pas dans la classe ou la structure dans laquelle la propriété est définie, ni dans une classe dérivée.
- L'instruction **Set** est marquée Friend (Visual Basic) et le code appelant ne figure pas dans l'assembly dans lequel la propriété est définie.

ID d'erreur : BC31102

Pour corriger cette erreur

- Si vous avez le contrôle du code source qui définit la propriété, vous devez déclarer la procédure **Set** avec le même niveau d'accès que la propriété proprement dite.
- Si vous n'avez pas le contrôle du code source qui définit la propriété, ou si vous devez limiter davantage le niveau d'accès de la procédure **Set** que celui de la propriété proprement dite, essayez de déplacer l'instruction qui définit la valeur de la propriété vers une zone de code qui offre un meilleur accès à la propriété.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Les instructions 'Select' ne sont pas valides dans la fenêtre Exécution

'Select' statements are not valid in the Immediate window

Les instructions **Select** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30718

Pour corriger cette erreur

- Supprimez les instructions **Select** du code de débogage.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

'Select Case' doit se terminer par un 'End Select' correspondant

'Select Case' must end with a matching 'End Select'

Une instruction **Select** ou **Select Case** apparaît sans instruction **End Select** correspondante. Une instruction **End Select** doit être utilisée pour terminer le bloc **Select**.

ID d'erreur : BC30095

Pour corriger cette erreur

1. Si ce bloc **Select** fait partie d'un ensemble de blocs **Select** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Ajoutez une instruction **End Select** à la fin du bloc **Select**.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

Attribut de sécurité '<nomattribut>' non valide :<erreur>

Security attribute '<attributename>' is not valid: <error>

Le Common Language Runtime n'accepte pas l'attribut dans un contexte de sécurité.

ID d'erreur : BC30128

Pour corriger cette erreur

1. Supprimez l'attribut de la déclaration ou remplacez-le par un attribut valide pour le contexte.
2. Examinez le message d'erreur cité et prenez les mesures qui s'imposent.

Voir aussi

Tâches

Comment : définir vos propres attributs

L'attribut de sécurité '<nomattribut>' ne peut pas être appliqué à un module

Security attribute '<attributename>' cannot be applied to a module

Le Common Language Runtime n'accepte pas l'attribut de sécurité sur un module.

ID d'erreur : BC30131

Pour corriger cette erreur

- Supprimez l'attribut de la déclaration.

Voir aussi

Tâches

Comment : définir vos propres attributs

Effets secondaires non valides lors de l'évaluation de l'expression dans ce contexte

Side effects not valid during expression evaluation in this context

Les instructions qui modifient l'état d'un objet ne sont pas autorisées dans ce contexte.

ID d'erreur : BC30700

Pour corriger cette erreur

- Modifiez le code de telle sorte que les instructions soient évaluées sans changer l'état d'autres éléments.

Voir aussi

Référence

Nothing (Visual Basic)

Une 'Sub New' partagée ne peut pas avoir de paramètre

Shared 'Sub New' cannot have any parameters

Vous avez inclus un ou des paramètres dans une instruction **Sub New** partagée, ce qui n'est pas valide.

ID d'erreur : BC30479

Pour corriger cette erreur

- Supprimez le ou les paramètres.

Voir aussi

Référence

Shared (Visual Basic)

Une 'Sub New' partagée ne peut pas être déclarée '<spécificateur>'

Shared 'Sub New' cannot be declared '<specifier>'

Vous avez tenté d'utiliser un spécificateur dans une déclaration **Shared Sub New**, ce qui n'est pas valide.

ID d'erreur : BC30480

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Shared (Visual Basic)
Sub, instruction (Visual Basic)

'Shared' ne peut pas être combiné avec '<spécificateur>' pour une déclaration de propriété

'Shared' cannot be combined with '<specifier>' on a property declaration

Vous avez tenté d'associer le modificateur **Shared** avec un spécificateur qui n'est pas valide dans une telle association de déclaration de propriété.

ID d'erreur : BC30502

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Shared (Visual Basic)

'Shared' ne peut pas être combiné avec '<spécificateur>' pour une déclaration de méthode

'Shared' cannot be combined with '<specifier>' on a method declaration

Vous avez tenté d'associer le modificateur **Shared** à un spécificateur, tel que **Overridable**, **NotOverridable** ou **MustOverride**, ce qui n'est pas valide dans une telle association de déclaration de méthode.

ID d'erreur : BC30501

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Shared (Visual Basic)
Overridable
NotOverridable
MustOverride

La propriété d'attribut 'Shared' '<champpropriété>' ne peut pas être la cible d'une assignation

'Shared' attribute property '<propertyfield>' cannot be the target of an assignment

Une tentative d'assignation d'une valeur à une propriété **ReadOnly** ou **Shared** dans un attribut a été effectuée.

ID d'erreur : BC31500

Pour corriger cette erreur

1. Supprimez l'instruction d'assignation de propriété.
2. Si vous utilisez des propriétés que vous avez développées, supprimez les modificateurs **ReadOnly** ou **Shared** de la propriété d'attribut.

Voir aussi

Référence

Shared (Visual Basic)

Autres ressources

Attributs en Visual Basic

Une instruction 'Set' doit se terminer par un 'End Set' correspondant

'Set' statement must end with a matching 'End Set'

Les procédures de propriété **Set** doivent se terminer par une instruction **End Set**.

ID d'erreur : BC30633

Pour corriger cette erreur

- Vérifiez que la procédure de propriété **Set** se termine par une construction **End Set**.

Voir aussi

Référence

Property, instruction

Concepts

Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

Le paramètre de 'Set' doit être du même type que la propriété conteneur

'Set' parameter must have the same type as the containing property

Le type du paramètre de la procédure de propriété **Set** est différent de la propriété à laquelle ce paramètre appartient.

ID d'erreur : BC31064

Pour corriger cette erreur

- Attribuez la valeur **Set** au type de données du paramètre de sorte qu'il corresponde au type de données de la propriété. Par exemple :

```
Class Class1
   ' Declare a local variable to hold the property value.
   Private Fval As Integer

   Property F() As Integer
      Get
         Return Fval
      End Get
      Set(ByVal Value As Integer)
         Fval = Value
      End Set
   End Property
End Class
```

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Concepts

Procédures Property

Propriétés et procédures Property

Le paramètre 'Set' ne peut pas être déclaré '<spécificateur>'

'Set' parameter cannot be declared '<specifier>'

Un modificateur non valide a été spécifié avec un paramètre dans une procédure de propriété **Set**.

ID d'erreur : BC31065

Pour corriger cette erreur

- Supprimez les modificateurs non valides, tels que **ByRef**, **ParamArray** et **Optional**.

Voir aussi

Tâches

Comment : ajouter des champs et des propriétés à une classe

Concepts

Procédures Property

Propriétés et procédures Property

Fichier source <nomfichier> spécifié plusieurs fois

Source file <filename> specified multiple times

Un fichier a été spécifié plusieurs fois.

ID d'erreur : BC2002

Pour corriger cette erreur

- Supprimez les spécificateurs de fichiers redondants.

Voir aussi

Référence

My.Computer.FileSystem, objet

L'accès spécifié '<NiveauAccès1>' pour '<NomTypePartiel>' ne correspond pas à l'accès '<NiveauAccès2>' spécifié pour l'un de ses autres types partiels

Specified access '<accesslevel1>' for '<partialtypename>' does not match the access '<accesslevel2>' specified on one of its other partial types

Une classe ou une structure est définie dans plusieurs déclarations partielles avec des spécifications de niveau d'accès incompatibles.

Lorsque vous divisez la définition d'une classe ou d'une structure en plusieurs déclarations partielles, le compilateur traite le type comme l'union de toutes ses déclarations partielles. Cela s'applique non seulement aux membres, mais également à l'implémentation, l'héritage et le niveau d'accès.

Vous ne pouvez pas associer des niveaux d'accès dans la définition d'une classe ou d'une structure. Même la combinaison **Protected Friend** est autorisée uniquement lorsque les mots clés sont contigus dans la même instruction de déclaration.

ID d'erreur : BC30925

Pour corriger cette erreur

- Déterminez le niveau d'accès de la classe, et supprimez les spécifications de niveau d'accès incompatibles.

Voir aussi

Référence

Partial (Visual Basic)
Class, instruction (Visual Basic)
Structure, instruction

Concepts

Niveaux d'accès dans Visual Basic
Classes : modèles d'objets

Autres ressources

Structures constituées à partir de vos propres types de données

Spécificateur dupliqué

Specifier is duplicated

Un mot clé apparaît plusieurs fois dans la même déclaration.

ID d'erreur : BC30178

Pour corriger cette erreur

- Supprimez de la déclaration les doublons du mot clé.

Voir aussi

Autres ressources

Éléments déclarés en Visual Basic

Les spécificateurs et les attributs ne sont pas valides sur les instructions 'Namespace'

Specifiers and attributes are not valid on 'Namespace' statements

Les espaces de noms sont utilisés pour permettre une organisation hiérarchique des blocs de programme et des éléments de programmation. Ils n'utilisent pas de modificateurs ou d'attributs.

ID d'erreur : BC32026

Pour corriger cette erreur

- Supprimez tous les blocs de mots clés et d'attributs de l'instruction **Namespace**.

Voir aussi

Référence

Namespace, instruction

Concepts

Espaces de noms dans Visual Basic

Vue d'ensemble des attributs dans Visual Basic

Application des attributs

Les spécificateurs et les attributs ne sont pas valides sur cette instruction

Specifiers and attributes are not valid on this statement

Une déclaration qui ne peut pas être modifiée, telle que **Inherits**, contient un ou plusieurs attributs ou un ou plusieurs mots clés, tels que **Public** ou **ReadOnly**. Seules les déclarations modifiables peuvent contenir de tels mots clés ou attributs.

ID d'erreur : BC30193

Pour corriger cette erreur

- Supprimez l'attribut ou le mot clé de la déclaration.

Voir aussi

Autres ressources

Mots clés (Visual Basic)
Attributs en Visual Basic

Les spécificateurs ne sont pas valides pour les méthodes 'AddHandler', 'RemoveHandler' et 'RaiseEvent'

Specifiers are not valid on 'AddHandler', 'RemoveHandler' and 'RaiseEvent' methods

Les déclarations de méthode **AddHandler**, **RemoveHandler** et **RaiseEvent** ne peuvent pas être modifiées avec des mots clés, tels que **Public** ou **ReadOnly**. Seules les déclarations modifiables peuvent contenir de tels mots clés.

ID d'erreur : BC31135

Pour corriger cette erreur

- Supprimez le mot clé de la déclaration de méthode.

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler
RaiseEvent

Autres ressources

Mots clés (Visual Basic)
Événements en Visual Basic

Les spécificateurs sont valides uniquement au début d'une déclaration

Specifiers valid only at the beginning of a declaration

Les mots clés spécifiant l'accessibilité et d'autres désignations doivent apparaître avant le mot clé de déclaration, tel que **Dim** ou **Function**.

ID d'erreur : BC30181

Pour corriger cette erreur

- Déplacez tous les spécificateurs entre la liste d'attributs (le cas échéant) et le mot clé de déclaration.

Voir aussi

Référence

Class, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Function, instruction (Visual Basic)
Property, instruction
Structure, instruction

L'instruction ne peut pas apparaître en dehors du corps d'une méthode

Statement cannot appear outside of a method body

Les instructions exécutables doivent être utilisées dans des procédures.

ID d'erreur : BC30689

Pour corriger cette erreur

- Déplacez l'instruction vers une procédure ou une sous-routine.

Voir aussi

Concepts

Procédures dans Visual Basic

Cette instruction ne peut pas apparaître dans le corps d'une méthode

Statement cannot appear within a method body

Une procédure contient une instruction non valide telle qu'une autre déclaration de procédure. Les déclarations de procédure ne peuvent pas être imbriquées.

ID d'erreur : BC30289

Pour corriger cette erreur

- Supprimez la déclaration de procédure ou déplacez-la à l'extérieur de la procédure englobante.

Voir aussi

Concepts

Procédures dans Visual Basic

Cette instruction ne peut pas apparaître dans le corps d'une propriété

Message d'erreur
Cette instruction ne peut pas apparaître dans le corps d'une propriété. Elle est interprétée comme la fin de la propriété.
Statement cannot appear within a property body. End of property assumed.

Une instruction inattendue a été rencontrée à l'intérieur d'une propriété.

ID d'erreur : BC30634

Pour corriger cette erreur

- Vérifiez que votre propriété utilise la syntaxe correcte et qu'elle se termine par une construction **End Property**.

Voir aussi

Référence
Property, instruction
Concepts
Modifications des procédures de propriétés pour les utilisateurs de Visual Basic 6.0

Cette instruction ne peut pas apparaître dans le corps d'un Enum

Message d'erreur
L'instruction ne peut pas apparaître dans le corps d'un Enum. Elle est interprétée comme la fin de l'Enum.
Statement cannot occur within an Enum body. End of Enum assumed.

Une construction de langage inattendue a été rencontrée. On suppose qu'une construction **End Enum** est manquante et que tout code source après ce point se trouve à l'extérieur de l'énumération.

ID d'erreur : BC30619

Pour corriger cette erreur

1. Vérifiez la syntaxe du code utilisée à l'intérieur de l'énumération.
2. Vérifiez que la définition d'interface se termine par une construction **End Enum**.

Voir aussi

Référence
Enum, instruction (Visual Basic)

Cette instruction ne peut pas apparaître dans le corps d'un événement

Message d'erreur
Cette instruction ne peut pas apparaître dans le corps d'un événement. Elle est interprétée comme la fin de l'événement.
Statement cannot appear within an event body. End of event assumed.

Une instruction non valide apparaît dans le corps d'un événement.

ID d'erreur : BC31112

Pour corriger cette erreur

- Ajoutez un **End Event** avant l'instruction.

Voir aussi

Tâches
Événements d'application, exemple
Référence
Event, instruction

L'instruction ne peut pas apparaître dans le corps d'une interface

Statement cannot appear within an interface body

La déclaration d'un membre d'interface contient une instruction qui met fin au membre du formulaire **End** *membername*.

Une interface définit uniquement la signature de ses membres. Par conséquent, les procédures et les propriétés définies dans une interface contiennent uniquement leur ligne initiale, qui spécifie le nom et la signature. Vous n'incluez pas de code, de déclarations internes ou d'instruction **End Function**, **End Property** ou **End Sub** dans l'interface.

ID d'erreur : BC30603

Pour corriger cette erreur

- Supprimez l'instruction **End** *membername* de la définition de l'interface.

Voir aussi

Référence

Interface, instruction (Visual Basic)
End (Visual Basic)

L'instruction ne peut pas apparaître dans le corps d'une interface (erreur Visual Basic)

Statement cannot appear within an interface body (Visual Basic Error)

Une construction de langage inattendue a été rencontrée. On suppose qu'une construction **End Interface** est manquante et que tout code source après ce point se trouve à l'extérieur de l'interface.

ID d'erreur : BC30604

Pour corriger cette erreur

1. Vérifiez la syntaxe du code utilisé à l'intérieur de la définition d'interface.
2. Vérifiez que la définition d'interface se termine par une construction **End Interface**.

Voir aussi

Référence

Interface, instruction (Visual Basic)

Autres ressources

Interfaces en Visual Basic

L'instruction ne peut pas terminer un bloc en dehors d'une instruction 'If' d'une ligne

Statement cannot end a block outside of a line 'If' statement

Une instruction **If** sur une seule ligne contient plusieurs instructions séparées par des deux-points (:), l'une est une instruction **End** pour un bloc de contrôle en dehors de la ligne unique **If**. Les instructions **If** sur une seule ligne n'utilisent pas l'instruction **End If**.

ID d'erreur : BC32005

Pour corriger cette erreur

- Déplacez cette instruction **If** en dehors du bloc de contrôle qui contient l'instruction **End If**.

Voir aussi

Référence

If...Then...Else, instruction (Visual Basic)

L'instruction ne déclare pas une méthode 'Get' ou 'Set'

Statement does not declare a 'Get' or 'Set' method

Votre instruction n'a pas pu fournir d'instruction de déclaration **Get** ou **Set** autour d'une procédure **Property**. Une propriété est définie comme un bloc de code délimité par les instructions **Property** et **End Property**. À l'intérieur de ce bloc, chaque procédure **Property** s'affiche sous forme de bloc interne délimité par une instruction de déclaration et une instruction de fin.

ID d'erreur : BC30576

Pour corriger cette erreur

- Spécifiez une instruction de déclaration **Get** ou **Set**.

Voir aussi

Concepts

Procédures Property

L'instruction ne déclare pas une méthode 'AddHandler', 'RemoveHandler' ou 'RaiseEvent'

Statement does not declare an 'AddHandler', 'RemoveHandler' or 'RaiseEvent' method

L'instruction ne fournit pas d'instruction de déclaration **AddHandler**, **RemoveHandler** ou instruction de déclaration **RaiseEvent** autour d'une procédure **Custom Event**. Une déclaration event personnalisée est un bloc de code délimité par les instructions **Custom Event** et **End Event**. À l'intérieur de ce bloc, chaque procédure **Custom Event** s'affiche sous forme de bloc interne délimité par une instruction de déclaration et une instruction **End**.

ID d'erreur : BC31113

Pour corriger cette erreur

- Spécifiez une instruction de déclaration **AddHandler**, **RemoveHandler** ou **RaiseEvent**.

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler
RaiseEvent

Autres ressources

Événements en Visual Basic

Instruction non valide dans un espace de noms

Statement is not valid in a namespace

L'instruction ne peut pas apparaître au niveau d'un espace de noms. Les seules déclarations autorisées au niveau de l'espace de noms sont les déclarations de module, d'interface, de classe, de délégué, d'énumération et de structure.

ID d'erreur : BC30001

Pour corriger cette erreur

- Déplacez l'instruction vers un emplacement situé à l'intérieur d'une définition de module, de classe, d'interface, de structure, d'énumération ou de délégué.

Voir aussi

Concepts

Portée dans Visual Basic
Espaces de noms dans Visual Basic

L'instruction n'est pas valide dans une interface

Statement is not valid in an interface

Une instruction non valide a été rencontrée dans l'interface.

ID d'erreur : BC31041

Pour corriger cette erreur

- Vérifiez que les instructions de l'interface définissent des éléments, mais qu'elles ne fournissent pas d'implémentation.

Voir aussi

Référence

Implements (Visual Basic)

Autres ressources

Interfaces en Visual Basic

Instruction non valide dans une méthode

Statement is not valid inside a method

L'instruction n'est pas valide à l'intérieur de **Sub**, de **Function**, d'une propriété **Get** ou d'une procédure de propriété **Set**. Certaines instructions peuvent être placées au niveau du module ou de la classe. D'autres, telles que **Option Strict**, doivent se trouver au niveau de l'espace de noms et précéder toutes les autres déclarations.

ID d'erreur : BC30024

Pour corriger cette erreur

- Supprimez l'instruction de la procédure.

Voir aussi

Référence

Sub, instruction (Visual Basic)
Function, instruction (Visual Basic)
Get, instruction
Set, instruction (Visual Basic)

L'instruction appelle de manière récursive le 'AddHandler' conteneur pour l'événement '<NomÉvénement>'

Statement recursively calls the containing 'AddHandler' for event '<eventname>'

Les instructions dans l'accesseur **AddHandler** d'une définition d'événement ne doivent pas référencer directement l'événement.

L'approche recommandée consiste à stocker la liste des gestionnaires de l'événement sous forme de champ privé dans la classe, la structure ou le module qui a défini l'événement. Pour plus d'informations, consultez
Comment : déclarer des événements qui évitent les blocages et
Comment : déclarer des événements qui économisent l'utilisation de la mémoire.

ID d'erreur : BC41998

Pour corriger cette erreur

- Réécrivez la définition de l'événement pour éviter toute récursivité.

Voir aussi

Tâches

Comment : déclarer des événements qui évitent les blocages
Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Référence

AddHandler
Event, instruction

Instructions et étiquettes non valides entre 'Select Case' et la première occurrence de 'Case'

Statements and labels are not valid between 'Select Case' and first 'Case'

Une instruction qui n'est pas un commentaire apparaît entre l'instruction ouvrante **Select** ou **Select Case** et sa première instruction **Case**.

ID d'erreur : BC30058

Pour corriger cette erreur

- Si l'instruction intermédiaire est un commentaire, faites-la précéder d'un délimiteur de commentaire (**'** ou **REM**). Sinon, déplacez ou supprimez l'instruction.

Voir aussi

Référence

Select...Case, instruction (Visual Basic)

Une variable locale statique '<nomvariable>' est déjà déclarée

Static local variable '<variablename>' is already declared

Une autre variable locale statique portant le même nom a déjà été déclarée.

ID d'erreur : BC31401

Pour corriger cette erreur

1. Supprimez toutes les déclarations locales statiques redondantes.
2. Attribuez à chaque variable locale statique un nom unique.

Voir aussi

Référence

Static (Visual Basic)

Une demande d'arrêt est en attente

Stop request is pending

Dans le débogueur Visual Studio, une expression spécifie un appel de procédure, mais une demande d'arrêt du thread est activée.

Le débogueur n'essaie pas d'appeler une procédure sur un thread qui n'est pas actif.

ID d'erreur : BC30946

Pour corriger cette erreur

- Si possible, déterminez la source de la demande d'arrêt du thread, afin que vous puissiez l'empêcher de se reproduire.
- Arrêtez et relancez le débogage.

Voir aussi

Tâches

Comment : démarrer l'exécution

Référence

Expressions dans Visual Basic

Concepts

Comment : arrêter le débogage ou l'exécution

Vue d'ensemble du code pas à pas

Autres ressources

Débogage dans Visual Studio

Les instructions 'Stop' ne sont pas valides dans la fenêtre Exécution

'Stop' statements are not valid in the Immediate window

Les instructions **Stop** et **End** interrompent l'exécution et ne sont pas autorisées dans un contexte de débogage.

ID d'erreur : BC30122

Pour corriger cette erreur

- N'émettez pas d'instruction **Stop** ou **End** dans la fenêtre **Exécution**.

Voir aussi

Référence

Fenêtre Exécution

Les constantes de chaîne doivent se terminer par un guillemet

String constants must end with a double quote

Les constantes de chaînes doivent commencer et se terminer par des guillemets.

ID d'erreur : BC30648

Pour corriger cette erreur

- Assurez-vous que le littéral de chaîne se termine avec un guillemet ("). Si vous collez des valeurs à partir d'autres éditeurs de texte, vérifiez que le caractère collé est un guillemet valide et non un caractère ressemblant à un guillemet comme les guillemets anglais (" ou ") ou deux guillemets simples ('').

Voir aussi

Autres ressources

Chaînes en Visual Basic

Constante de chaîne attendue

String constant expected

Un caractère autre qu'un guillemet double (") apparaît lorsque le contexte requiert une constante de chaîne. Une des causes possibles est l'absence de guillemets dans la chaîne.

ID d'erreur : BC30217

Pour corriger cette erreur

1. Vérifiez que la chaîne est correctement placée entre des guillemets doubles.
2. Fournissez une constante de chaîne.

Voir aussi

Concepts

Constantes et types de données littérales

'Sub' ou 'Function' attendu après 'Delegate'

'Sub' or 'Function' expected after 'Delegate'

Une instruction **Delegate** ne spécifie pas de procédure **Sub** ou **Function**. Le mot clé **Sub** ou **Function** doit suivre immédiatement le mot clé **Delegate**.

ID d'erreur : BC30278

Pour corriger cette erreur

1. Ajoutez le mot clé **Sub** ou **Function** immédiatement après **Delegate**.
2. Spécifiez un nom de procédure, une liste d'arguments et un type de retour, selon le cas.

Voir aussi

Référence

Delegate, instruction

Concepts

Procédures dans Visual Basic

'Sub' ou 'Function' attendu

'Sub' or 'Function' expected

Une instruction **Declare** ne spécifie ni **Sub**, ni **Function**. **Declare** identifie une procédure externe et doit spécifier le type de procédure.

ID d'erreur : BC30215

Pour corriger cette erreur

- Ajoutez le mot clé **Sub** ou **Function** à l'instruction **Declare**.

Voir aussi

Référence

Declare, instruction

'Sub New' ne peut pas implémenter des membres d'interface

'Sub New' cannot implement interface members

Sub New est un constructeur et ne peut pas implémenter de membres.

ID d'erreur : BC31042

Pour corriger cette erreur

- Supprimez les instructions **Implements** des procédures **Sub New**.

Voir aussi

Référence

Implements (Visual Basic)

Autres ressources

Interfaces en Visual Basic

'Sub New' ne peut pas gérer d'événements

'Sub New' cannot handle events

Vous avez tenté d'associer **Sub New** à **Handles**, ce qui n'est pas valide. Utilisez le mot clé **Handles** à la fin d'une déclaration de procédure pour que celle-ci gère les événements déclenchés par une variable objet déclarée à l'aide du mot clé **WithEvents**.

ID d'erreur : BC30497

Pour corriger cette erreur

- N'utilisez pas **New** dans ce contexte.

Voir aussi

Référence

Handles
Dim, instruction (Visual Basic)
WithEvents

'Sub New' ne peut pas être déclaré 'Overrides'

'Sub New' cannot be declared 'Overrides'

Un constructeur indique qu'il se substitue à un constructeur hérité. Les constructeurs ne peuvent pas être substitués.

ID d'erreur : BC30283

Pour corriger cette erreur

- Supprimez le mot clé **Overrides** de la déclaration **Sub**.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

Concepts

Substitution de propriétés et de méthodes

'Sub New' ne peut pas être déclaré dans une interface

'Sub New' cannot be declared in an interface

Vous avez tenté de déclarer **Sub New** dans une interface. Dans la mesure où il s'agit d'un constructeur, **Sub New** ne peut s'exécuter qu'une seule fois lorsqu'une classe est créée. Il ne peut être appelé explicitement qu'à partir de la première ligne de code dans un autre constructeur dans la même classe ou une classe dérivée.

ID d'erreur : BC30363

Pour corriger cette erreur

- Supprimez le constructeur **Sub New** ou déplacez-le vers un emplacement mieux adapté dans votre code.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

'Sub New' ne peut pas être déclaré '<modificateur>'

'Sub New' cannot be declared '<modifier>'

Vous avez tenté de déclarer un **Sub New** avec un modificateur, tel que **Overrides**, ce qui n'est pas valide.

ID d'erreur : BC30364

Pour corriger cette erreur

- Supprimez le modificateur.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

'Sub Main' est introuvable dans '<nom>'

'Sub Main' was not found in '<name>'

Sub Main est manquante ou un emplacement incorrect a été spécifié.

ID d'erreur : BC30420

Pour corriger cette erreur

1. Fournissez l'instruction **Sub Main** manquante ou, si elle existe, déplacez-la vers l'emplacement approprié dans le code. Pour plus d'informations sur **Sub Main**, consultez Procédure Main dans Visual Basic.
2. Spécifiez l'emplacement de l'objet de démarrage du projet dans la zone **Formulaire de démarrage** du **Concepteur de projets**.

Pour plus d'informations, consultez Comment : modifier l'objet de démarrage pour une application et Comment : modifier l'objet de démarrage pour une application (Visual Basic) et Comment : modifier l'objet de démarrage pour une application (Visual Basic).

Voir aussi

Référence

Sub, instruction (Visual Basic)

Concepts

Procédure Main dans Visual Basic

'Sub Main' est déclaré plusieurs fois dans '<espacedenoms>' : <message>

'Sub Main' is declared more than once in '<namespace>': <message>

Sub Main ne peut être déclaré qu'une seule fois dans un espace de noms.

ID d'erreur : BC30738

Pour corriger cette erreur

- Vérifiez qu'il n'existe qu'une seule procédure **Sub Main** dans votre projet.

Voir aussi

Concepts

Procédures dans Visual Basic

Les structures ne peuvent pas utiliser d'instructions 'Inherits'

Structures cannot have 'Inherits' statements

À la différence des classes, les structures ne prennent pas en charge l'héritage.

ID d'erreur : BC30628

Pour corriger cette erreur

- Supprimez l'instruction **Inherits** de la structure.
- Recréez votre application pour utiliser plutôt des classes.

Voir aussi

Concepts

Structures et classes

Autres ressources

Structures constituées à partir de vos propres types de données

Les structures ne peuvent pas déclarer un 'Sub New' non partagé sans paramètre

Structures cannot declare a non-shared 'Sub New' with no parameters

Les constructeurs **Sub New** déclarés dans des structures doivent accepter les arguments ou être déclarés avec le modificateur **Shared**.

ID d'erreur : BC30629

Pour corriger cette erreur

- Modifiez le constructeur **Sub New** afin qu'il accepte les arguments.
- Appliquez le modificateur **Shared** pour que le constructeur soit partagé.

Voir aussi

Référence

Structure, instruction

Autres ressources

Structures constituées à partir de vos propres types de données

L'instruction 'Structure' doit se terminer par un 'End Structure' correspondant

'Structure' statement must end with a matching 'End Structure'

Les structures commencent par une instruction **Structure** et se terminent par une construction **End Structure**.

ID d'erreur : BC30624

Pour corriger cette erreur

- Vérifiez que le bloc de structure se termine par une instruction **End Structure**.

Voir aussi

Concepts

Structures et classes

Autres ressources

Structures constituées à partir de vos propres types de données

Une structure dans une interface ne peut pas être déclarée '<spécificateur>'

Structure in an interface cannot be declared '<specifier>'

Un modificateur non valide a été spécifié avec une structure déclarée dans une interface.

ID d'erreur : BC31071

Pour corriger cette erreur

- Supprimez les modificateurs, tels que **Public**, **Private**, **Shared**, **Friend**, **Protected**, **Protected Friend** ou **Overridable**.

Voir aussi

Référence

Structure, instruction

Autres ressources

Interfaces en Visual Basic

La contrainte 'Structure' ne peut être spécifiée plusieurs fois pour le même paramètre de type

'Structure' constraint cannot be specified multiple times for the same type parameter

Une liste de contraintes inclut plusieurs fois la contrainte Structure (Visual Basic).

Une liste de contraintes d'un paramètre de type peut spécifier que l'argument de type passé à ce paramètre de type doit être un type valeur (avec la contrainte **Structure**) ou un type référence (avec la contrainte Class (Visual Basic)). Vous ne pouvez pas spécifier les deux contraintes pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32102

Pour corriger cette erreur

- Supprimez tous les mots clés **Structure** redondants. Vous ne devez en avoir qu'un seul dans la liste de contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

Une contrainte 'Structure' et une contrainte de type classe spécifique ne peuvent pas être combinées

'Structure' constraint and a specific class type constraint cannot be combined

Une liste de contraintes inclut la contrainte Structure (Visual Basic) et le nom d'une classe définie.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Vous pouvez spécifier les éléments requis suivants selon n'importe quelle combinaison :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit hériter d'une classe au plus
- L'argument de type doit exposer un constructeur sans paramètre accessible par le code de création (contrainte **New** à inclure)

Si vous n'incluez aucune classe ou interface spécifique dans la liste de contraintes, vous pouvez imposer une condition plus générale en spécifiant l'un des éléments suivants :

- l'argument de type doit être un type valeur (contrainte **Structure** à inclure) ;
- l'argument de type doit être un type de référence (contrainte **Class** à inclure) ;

Vous ne pouvez pas spécifier **Structure** et **Class** pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32108

Pour corriger cette erreur

- Si vous souhaitez que l'argument de type soit un type valeur, supprimez le nom de classe de la liste de contraintes.
- Si vous souhaitez que l'argument de type hérite du nom de classe spécifié, supprimez le mot clé **Structure** de la liste de contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic
Types valeur et types référence

La structure '<nomstructure>' doit contenir au moins une déclaration de variable membre d'instance ou une déclaration Event

Structure '<structurename>' must contain at least one instance member variable or Event declaration

Une structure vide sur le plan fonctionnel apparaît. Au moins un membre de variable doit être déclaré entre l'instruction **Structure** et l'instruction **End Structure**

ID d'erreur : BC30281

Pour corriger cette erreur

- Ajoutez une instruction **Dim** ou une instruction **Event** à la structure.

Voir aussi

Référence

Structure, instruction
Dim, instruction (Visual Basic)
Event, instruction

La structure '<nomstructure>' doit contenir au moins une variable membre d'instance ou au moins une déclaration d'événement d'instance non marquée comme 'Custom'.

Structure '<structurename>' must contain at least one instance member variable or at least one instance event declaration not marked 'Custom'

Une définition de structure n'inclut pas de variables non partagées ou d'événements non personnalisés non partagés.

Chaque structure doit avoir une variable ou un événement qui s'applique à chaque instance spécifique (non partagée) et non à toutes les instances collectivement (Shared (Visual Basic)). Les constantes, propriétés et procédures non partagées ne répondent pas à cette condition. Par ailleurs, s'il n'y a aucune variable non partagée et qu'un seul événement non partagé, cet événement ne peut pas être un événement Custom.

ID d'erreur : BC30941

Pour corriger cette erreur

- Définissez au moins une variable ou un événement qui n'est pas **Shared**. Si vous ne définissez qu'un événement, il doit être non personnalisé et non partagé.

Voir aussi

Tâches

Comment : déclarer une structure

Référence

Structure, instruction

Autres ressources

Structures constituées à partir de vos propres types de données

La structure '<NomStructure>' ne peut pas contenir une instance d'elle-même : <erreur>

Structure '<structurename>' cannot contain an instance of itself: <error>

Une structure déclare une variable et l'initialise avec une instance d'elle-même.

Une structure peut contenir des instances d'autres structures, mais pas une instance interne d'elle-même. Toute tentative entraîne une récurrence infinie.

ID d'erreur : BC30294

Pour corriger cette erreur

1. Vérifiez l'orthographe de l'expression d'initialisation dans l'instruction de déclaration.
2. Si vous comptez créer une autre instance de la même structure, vous devez la déclarer et la créer à l'extérieur de la structure.

Voir aussi

Référence

Structure, instruction

Autres ressources

Structures constituées à partir de vos propres types de données

La structure '<nomstructure>' ne peut pas être indexée, car elle n'a pas de propriété par défaut

Structure '<structurename>' cannot be indexed because it has no default property

Une tentative d'utilisation de la syntaxe de la propriété par défaut avec une structure n'ayant pas de propriété par défaut a été effectuée.

ID d'erreur : BC30690

Pour corriger cette erreur

- Utilisez la syntaxe de la propriété standard pour accéder aux propriétés de la structure.

Voir aussi

Concepts

Modifications des propriétés par défaut pour les utilisateurs de Visual Basic 6.0
Structures et autres éléments de programmation
Propriétés par défaut

Erreur de syntaxe dans l'expression de compilation conditionnelle

Syntax error in conditional compilation expression

L'expression de compilation conditionnelle génère une erreur de syntaxe non spécifique.

ID d'erreur : BC31427

Pour corriger cette erreur

1. Consultez la documentation relative à une compilation conditionnelle et aux mots clés utilisés dans l'expression. Comparez l'exemple de code qui se trouve dans la documentation avec la ligne source générant cette erreur.
2. Si vous ne pouvez pas identifier la cause de l'erreur, rassemblez les informations ayant trait aux circonstances et informez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Compilation conditionnelle (Visual Basic)

Erreur de syntaxe dans l'opérateur de cast ; deux arguments séparés par une virgule sont requis

Syntax error in cast operator; two arguments separated by comma are required

Une expression utilise la fonction de conversion **CType** ou le mot clé de conversion **DirectCast** ou **TryCast**, mais ne fournit qu'un seul argument.

CType, **DirectCast** et **TryCast** nécessitent deux arguments. Le premier argument est une expression à convertir et le second argument est le type de données ou de classe de conversion.

ID d'erreur : BC30944

Pour corriger cette erreur

- Fournissez les deux arguments selon les besoins de conversion.
- Si vous comptez utiliser l'un des Fonctions de conversion de types de données spécifiques, par exemple **CString**, vous devez utiliser ce nom de fonction au lieu de **CType**. Un seul argument est requis.

Voir aussi

Référence

CType, fonction
DirectCast
TryCast
Fonctions de conversion de types de données

Erreur de syntaxe (erreur Visual Basic)

Syntax error (Visual Basic Error)

La ligne du code source génère une erreur de syntaxe non spécifique.

ID d'erreur : BC30035

Pour corriger cette erreur

1. Consultez la documentation relative à chaque mot clé utilisé dans la ligne du code source. Comparez l'exemple de code qui se trouve dans la documentation avec la ligne source générant cette erreur.
2. Si vous ne pouvez pas identifier la cause de l'erreur, rassemblez les informations ayant trait aux circonstances et informez les services de Support Technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Les instructions 'SyncLock' ne sont pas valides dans la fenêtre Exécution

'SyncLock' statements are not valid in the Immediate window

L'instruction **SyncLock** synchronise les threads et n'est pas autorisée dans un contexte de débogage.

ID d'erreur : BC30135

Pour corriger cette erreur

- N'émettez pas d'instruction **SyncLock** dans la fenêtre **Exécution**.

Voir aussi

Référence

Fenêtre Exécution
SyncLock, instruction

L'instruction 'SyncLock' doit se terminer par un 'End SyncLock' correspondant

'SyncLock' statement must end with a matching 'End SyncLock'

Les blocs **SyncLock** commencent par le mot clé **SyncLock** et se terminent par la construction **End SyncLock**.

ID d'erreur : BC30675

Pour corriger cette erreur

- Vérifiez que le bloc **SyncLock** se termine par une construction **End SyncLock**.

Voir aussi

Référence

SyncLock, instruction

L'opérande 'SyncLock' ne peut pas être de type '<nomtype>', car '<nomtype>' n'est pas un type référence

'SyncLock' operand cannot be of type '<typename>' because '<typename>' is not a reference type

L'instruction **SyncLock** permet de synchroniser des instructions sur une seule expression, ce qui garantit que plusieurs threads n'exécutent pas les mêmes instructions simultanément. Le type d'expression dans une instruction **SyncLock** doit être un type référence, par exemple une classe, un module, une interface, un tableau ou un délégué.

ID d'erreur : BC30582

Pour corriger cette erreur

- Modifiez le type en un type référence approprié.

Voir aussi

Référence

SyncLock, instruction

Autres ressources

Multithreading dans Visual Basic

Une demande d'interruption est en attente

Suspend request is pending

Dans le débogueur Visual Studio, une expression spécifie un appel de procédure, mais une demande de suspension du thread est activée.

Le débogueur n'essaie pas d'appeler une procédure sur un thread qui n'est pas actif.

ID d'erreur : BC30947

Pour corriger cette erreur

- Si possible, déterminez la source de la demande de suspension du thread, afin que vous puissiez l'empêcher de se reproduire.
- Arrêtez et relancez le débogage.

Voir aussi

Tâches

Comment : démarrer l'exécution

Référence

Expressions dans Visual Basic

Concepts

Comment : arrêter le débogage ou l'exécution

Vue d'ensemble du code pas à pas

Autres ressources

Débogage dans Visual Studio

'System.Void' ne peut être utilisé que dans une expression GetType

'System.Void' can only be used in a GetType expression

Une expression dans une instruction d'assignation ou une déclaration utilise Void comme type de variable, de paramètre de procédure, de retour de fonction, ou d'argument de type.

La structure **Void** est un type spécialisé utilisé en interne par le .NET Framework et en particulier par Visual C# et Visual C++. Elle représente un type valeur de retour pour une méthode qui ne retourne pas de valeur. Visual Basic utilise une procédure **Sub** lorsque aucune valeur n'est retournée et une procédure **Function** lorsqu'une valeur est retournée.

Vous pouvez tester une variable référence à l'aide de l'opérateur GetType pour déterminer si son type d'exécution a la valeur **Void**, mais vous ne pouvez pas utiliser **Void** dans un autre contexte.

ID d'erreur : BC31422

Pour corriger cette erreur

1. Si vous souhaitez comparer le type d'exécution d'une variable à **Void**, utilisez l'opérateur **GetType**.
2. À moins que vous ayez une raison particulière de comparer un type d'exécution à **Void**, supprimez entièrement la référence à ce type.

Voir aussi

Référence

GetType, opérateur
Void

Impossible d'appliquer à la fois 'System.STAThreadAttribute' et 'System.MTAThreadAttribute' à la même méthode

'System.STAThreadAttribute' and 'System.MTAThreadAttribute' cannot both be applied to the same method

Les attributs **System.STAThreadAttribute** et **System.MTAThreadAttribute** s'excluent mutuellement.

ID d'erreur : BC31512

Pour corriger cette erreur

- Appliquez **System.MTAThreadAttribute** ou **System.STAThreadAttribute**, mais pas les deux.

Voir aussi

Référence

STAThreadAttribute Class

MTAThreadAttribute Class

Autres ressources

Attributs en Visual Basic

Impossible d'appliquer à la fois 'System.STAThreadAttribute' et 'System.MTAThreadAttribute' à '|1'

'System.STAThreadAttribute' and 'System.MTAThreadAttribute' cannot both be applied to '|1'

Les attributs **System.STAThreadAttribute** et **System.MTAThreadAttribute** s'excluent mutuellement.

ID d'erreur : BC31513

Pour corriger cette erreur

- Appliquez **System.MTAThreadAttribute** ou **System.STAThreadAttribute**, mais pas les deux.

Voir aussi

Référence

STAThreadAttribute Class

MTAThreadAttribute Class

Autres ressources

Attributs en Visual Basic

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué aux méthodes d'interface

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to interface methods

Une procédure est définie à l'intérieur d'une interface, mais la définition de la procédure applique l'DllImportAttribute.

Le Common Language Runtime (CLR) reconnaît cet attribut et sa propriété EntryPoint comme désignant une procédure de remplacement définie dans une bibliothèque de liens dynamiques (DLL) non managée en dehors de .NET Framework. Lorsque le code appelle la procédure à laquelle l'**DllImportAttribute** est appliqué, le Common Language Runtime appelle plutôt la procédure non managée désignée.

Étant donné que la définition d'une procédure à l'intérieur d'une interface n'inclut aucune implémentation, il ne peut pas interagir avec les plates-formes non managées à l'extérieur de .NET Framework.

ID d'erreur : BC31530

Pour corriger cette erreur

- Supprimez l'**DllImportAttribute** de la définition de cette procédure.

Voir aussi

Référence

Interface, instruction (Visual Basic)
DllImportAttribute

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué à une méthode d'instance

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to instance methods

Une procédure non partagée est déclarée avec l'DllImportAttribute.

Le Common Language Runtime (CLR) reconnaît cet attribut et sa propriété EntryPoint comme désignant une procédure de remplacement définie dans une bibliothèque de liens dynamiques (DLL) non managée en dehors de .NET Framework. Lorsque le code appelle la procédure à laquelle l'**DllImportAttribute** est appliqué, le Common Language Runtime appelle plutôt la procédure non managée désignée.

Étant donné que les plates-formes non managées en dehors de .NET Framework ne prennent pas en charge les procédures non partagées de la même manière que .NET Framework, vous ne pouvez pas interagir avec elles en utilisant des procédures non partagées.

ID d'erreur : BC31529

Pour corriger cette erreur

- S'il n'est pas nécessaire que la procédure s'applique individuellement à chaque instance de sa classe ou de sa structure, déclarez-la comme **Shared**.
- Si la procédure ne peut pas être **Shared**, supprimez l'**DllImportAttribute** de la déclaration de cette procédure.

Voir aussi

Référence

Shared (Visual Basic)
DllImportAttribute

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué à une méthode 'AddHandler', 'RemoveHandler' ou 'RaiseEvent'

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to 'AddHandler', 'RemoveHandler', or 'RaiseEvent' methods

L'attribut **DllImportAttribute** a été appliqué à une déclaration de méthode **AddHandler**, **RemoveHandler** ou **RaiseEvent**. Cet attribut peut seulement être appliqué à un **Function** ou **Sub** vide.

ID d'erreur : BC31531

Pour corriger cette erreur

- Supprimez l'attribut **DllImportAttribute** de la déclaration de méthode.

Voir aussi

Référence

Event, instruction
AddHandler
RemoveHandler
RaiseEvent
DllImportAttribute

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué à un Sub, Function ou Operator avec un corps non vide

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to a Sub, Function, or Operator with a non-empty body

L'attribut **DllImportAttribute** a été appliqué à un **Sub**, **Function** ou **Operator** qui n'est pas vide.

ID d'erreur : BC31522

Pour corriger cette erreur

- Supprimez tout le code du **Sub**, **Function** ou **Operator** pour utiliser cet attribut.

Voir aussi

Référence

DllImportAttribute Class
Declare, instruction

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué à une méthode générique ou imbriquée dans un type générique

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to a method that is generic or nested in a generic type

Une procédure est déclarée avec DllImportAttribute, mais la procédure est générique ou elle est contenue dans une classe ou une structure générique.

Le Common Language Runtime (CLR) reconnaît cet attribut et sa propriété EntryPoint comme désignant une procédure de remplacement définie dans une bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) non managée à l'extérieur du .NET Framework. Lorsque le code appelle la procédure à laquelle **DllImportAttribute** est appliqué, le Common Language Runtime appelle à la place la procédure non managée désignée.

Étant donné que les plates-formes non managées à l'extérieur du .NET Framework ne reconnaissent pas les types génériques, vous ne pouvez pas interagir avec eux à l'aide de types génériques.

ID d'erreur : BC31526

Pour corriger cette erreur

- Si ni la procédure ni son conteneur ne doivent être génériques, supprimez les clauses **Of** pour qu'elles ne soient pas génériques.
- Si la procédure ou son conteneur doit être générique, supprimez **DllImportAttribute** de la déclaration de cette procédure.

Voir aussi

Référence

DllImportAttribute

Concepts

Types génériques en Visual Basic

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué à un 'Get' ou 'Set'

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to a 'Get' or 'Set'

Vous avez appliqué l'attribut **DllImportAttribute** à une procédure de propriété **Get** ou **Set**.

ID d'erreur : BC31524

Pour corriger cette erreur

- Supprimez **DllImportAttribute** des procédures de propriété **Get** et **Set**.

Voir aussi

Référence

DllImportAttribute Class

Concepts

Procédures Property

'System.Runtime.InteropServices.DllImportAttribute' ne peut pas être appliqué à un 'Declare'

'System.Runtime.InteropServices.DllImportAttribute' cannot be applied to a 'Declare'

Vous avez appliqué l'attribut **DllImportAttribute** à une fonction **Declare**. Cet attribut peut seulement être appliqué à un **Function** ou **Sub** vide.

ID d'erreur : BC31523

Pour corriger cette erreur

- Supprimez l'attribut **DllImportAttribute** de l'instruction **Declare**.

Voir aussi

Référence

DllImportAttribute Class
Declare, instruction

Impossible d'appliquer 'System.Runtime.InteropServices.DispIdAttribute' à '<nomtype>', car 'Microsoft.VisualBasic.ComClassAttribute' réserve zéro à la propriété par défaut

'System.Runtime.InteropServices.DispIdAttribute' value cannot be applied to '<typename>' because 'Microsoft.VisualBasic.ComClassAttribute' reserves zero for the default property

Un bloc d'attributs DispIdAttribute spécifie une valeur DISPID égale à 0, qui est réservée par **COMClassAttribute** pour représenter la propriété par défaut de la classe à laquelle elle s'applique.

L'identificateur de dispatch (DISPID) est utilisé dans COM comme un argument de la méthode **IDispatch:Invoke** pour accéder aux propriétés et aux méthodes exposées par un objet COM.

ID d'erreur : BC32505

Pour corriger cette erreur

- Spécifiez une valeur DISPID supérieure à zéro dans **DispIdAttribute**.

Voir aussi

Référence

ComClassAttribute, classe

DispIdAttribute

Concepts

Attributs utilisés dans Visual Basic

Application des attributs

Impossible d'appliquer 'System.Runtime.InteropServices.DispIdAttribute' à '<nomtype>', car 'Microsoft.VisualBasic.ComClassAttribute' réserve les valeurs inférieures à zéro

'System.Runtime.InteropServices.DispIdAttribute' value cannot be applied to '<typename>' because 'Microsoft.VisualBasic.ComClassAttribute' reserves values less than zero

Un bloc d'attributs DispIdAttribute spécifie une valeur DISPID inférieure à 0, qui est réservée par **COMClassAttribute** pour des fonctions spéciales sur la classe à laquelle elle s'applique.

L'identificateur de dispatch (DISPID) est utilisé dans COM comme un argument de la méthode **IDispatch:Invoke** pour accéder aux propriétés et aux méthodes exposées par un objet COM.

ID d'erreur : BC32506

Pour corriger cette erreur

- Spécifiez une valeur DISPID supérieure à zéro dans **DispIdAttribute**.

Voir aussi

Référence

ComClassAttribute, classe

DispIdAttribute

Concepts

Attributs utilisés dans Visual Basic

Application des attributs

'System.ObsoleteAttribute' ne peut pas être appliqué aux définitions 'AddHandler', 'RemoveHandler' ou 'RaiseEvent'

Message d'erreur
''System.ObsoleteAttribute' ne peut pas être appliqué aux définitions 'AddHandler', 'RemoveHandler' ou 'RaiseEvent'. Si nécessaire, appliquez directement l'attribut à l'événement.
'System.ObsoleteAttribute' cannot be applied to the 'AddHandler', 'RemoveHandler', or 'RaiseEvent' definitions. If required, apply the attribute directly to the event.

Un événement personnalisé applique ObsoleteAttribute à sa procédure **AddHandler**, **RemoveHandler** ou **RaiseEvent**.

ObsoleteAttribute marque un élément de programmation comme n'étant plus utilisé et informe les utilisateurs que l'élément sera supprimé dans les prochaines versions du produit.

Cela n'a pas de sens que certaines parties d'un événement personnalisé soient toujours utilisées tandis que d'autres ne le sont plus.

ID d'erreur : BC31142

Pour corriger cette erreur

- Supprimez **ObsoleteAttribute** de la déclaration de procédure et appliquez-le à l'ensemble de la déclaration de l'événement.

Voir aussi

Référence
Event, instruction
AddHandler, instruction
RemoveHandler, instruction
RaiseEvent, instruction
ObsoleteAttribute

'System.Nullable' ne satisfait pas la contrainte 'Structure' pour le paramètre de type '<NomParamètreDeType>'

'System.Nullable' does not satisfy the 'Structure' constraint for type parameter '<typeparametername>'

Un type générique est appelé en passant un argument de type de Nullable à un paramètre de type avec une contrainte **Structure**.

Le Common Language Runtime (CLR) interdit spécifiquement la structure **Nullable** en tant qu'argument de type à lui-même. Bien qu'il s'agisse d'une structure et qu'elle pourrait satisfaire une contrainte **Structure**, son utilisation récursive peut entraîner des constructions maladroites telles que `Nullable(Of Nullable(Of Nullable))`.

ID d'erreur : BC32115

Pour corriger cette erreur

- Supprimez la contrainte **Structure** du paramètre de type, ou remplacez l'argument de type par un type valeur autre que **Nullable**.

Voir aussi

Référence

Structure (Visual Basic)

Nullable

Concepts

Types génériques en Visual Basic

System.Diagnostics.DebuggerHiddenAttribute n'affecte pas 'Get' ou 'Set' lorsqu'il est appliqué à la définition Property

Message d'erreur
System.Diagnostics.DebuggerHiddenAttribute n'affecte pas 'Get' ou 'Set' lorsqu'il est appliqué à la définition Property. Appliquez l'attribut directement aux procédures 'Get' et 'Set', selon le cas.
System.Diagnostics.DebuggerHiddenAttribute does not affect 'Get' or 'Set' when applied to the Property definition. Apply the attribute directly to the 'Get' and 'Set' procedures as appropriate.

DebuggerHiddenAttribute est appliqué à une déclaration de propriété.

Le code source peut appliquer **DebuggerHiddenAttribute** à une procédure. Ainsi, le débogueur Visual Studio 2005 ne s'arrête pas à l'intérieur de la procédure et n'autorise pas la définition de points d'arrêt dans la procédure.

Bien que vous puissiez appliquer **DebuggerHiddenAttribute** à une propriété, cela n'a aucun effet. Pour obtenir l'effet escompté, vous devez l'appliquer à la procédure **Get** ou **Set** de la propriété.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40051

Pour corriger cette erreur

- Supprimez **DebuggerHiddenAttribute** de la déclaration de propriété et appliquez-le à la procédure **Get** ou **Set** de la propriété, selon le cas.

Voir aussi

Référence
Property, instruction
Get, instruction
Set, instruction (Visual Basic)
DebuggerHiddenAttribute
Concepts
Procédures Property

System.CLSCompliantAttribute ne peut pas être appliqué à la propriété 'Get'/'Set'

System.CLSCompliantAttribute cannot be applied to property 'Get'/'Set'

Une définition de propriété applique l'attribut CLSCompliantAttribute à son instruction **Get** ou **Set**.

Pour qu'une propriété soit conforme au Spécification CLS (Common Language Specification) (CLS), la propriété entière doit être marquée comme `<CLSCompliant(True)>`. Vous devez appliquer **CLSCompliantAttribute** à l'instruction Property, et non à l'instruction **Get** ou **Set**.

Lorsque vous appliquez **CLSCompliantAttribute** à un élément de programmation, vous affectez au paramètre **isCompliant** de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40043

Pour corriger cette erreur

- Supprimez **CLSCompliantAttribute** de l'instruction **Get** ou **Set**.
- Si la propriété doit être conforme CLS, marquez l'instruction **Property** comme `<CLSCompliant(True)>`.

Voir aussi

Référence

Get, instruction

Set, instruction (Visual Basic)

Concepts

Écriture d'un code conforme CLS

Le fichier '<nomfichier>' n'est pas un fichier texte

The file '<filename>' is not a text file

L'écriture n'est pas autorisée dans le fichier spécifié car il s'agit d'un fichier binaire.

ID d'erreur : BC2015

Pour corriger cette erreur

- Vérifiez que le nom et le chemin d'accès du fichier fournis sont corrects.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem.WriteAllText, méthode

My.Computer.FileSystem.WriteAllBytes, méthode

Autres ressources

Écriture dans des fichiers en Visual Basic

La référence de l'assembly par défaut '<référence>' a déjà été automatiquement ajoutée ; ignoré

The default assembly reference '<reference>' has already been automatically added; ignored

Une tentative a été réalisée pour ajouter la référence de l'assembly par défaut, mais l'assembly ayant déjà été ajouté, la tentative est ignorée.

ID d'erreur : BC2024

Pour corriger cette erreur

- Supprimez le code superflu.

Voir aussi

Concepts

Assemblys

Le ConnectionTimeout doit être supérieur à 0

The ConnectionTimeout must be greater than 0

Lors du téléchargement de fichiers avec l'objet My.Computer.Network, vous devez spécifier une valeur *connectionTimeout* supérieure à `0`.

Pour corriger cette erreur

- Entrez une valeur *connectionTimeout* supérieure à `0`.

Voir aussi

Tâches

Comment : transférer un fichier dans Visual Basic

Comment : télécharger un fichier dans Visual Basic

Référence

My.Computer.Network.UploadFile, méthode

My.Computer.Network.DownloadFile, méthode

Autres ressources

Opérations de réseau dans le .NET Framework avec Visual Basic

Le mot clé '<MotClé>' est utilisé pour surcharger des membres hérités ; n'utilisez pas le mot clé '<MotClé>' lors de la surcharge de 'Sub New'

The '<keyword>' keyword is used to overload inherited members; do not use the '<keyword>' keyword when overloading 'Sub New'

Un constructeur est déclaré avec le mot clé **Overloads**.

Visual Basic ne prend pas en charge l'héritage ou la surcharge de constructeurs.

ID d'erreur : BC32040

Pour corriger cette erreur

- Supprimez le mot clé **Overloads** de toutes les déclarations du constructeur.

Voir aussi

Référence

Overloads
Utilisation de constructeurs et de destructeurs

la valeur '<valeur>' n'est pas valide pour l'option '<nomoption>'

the value '<value>' is invalid for option '<optionname>'

Une valeur non valide a été fournie pour une option de ligne de commande.

ID d'erreur : BC2014

Pour corriger cette erreur

- Vérifiez les valeurs pour rechercher celle qui n'est pas valide.

Voir aussi

Référence

Switch, fonction

Le type '<NomType>' ne peut pas être un type d'élément de tableau, un type de retour, un type de champ, un type d'argument générique, un type de paramètre 'ByRef' ou le type d'une expression convertie en 'Object' ou 'ValueType'

The type '<typename>' cannot be an array element type, return type, field type, generics argument type, 'ByRef' parameter type or the type of an expression converted to 'Object' or 'ValueType'

Une expression déclare une variable, un paramètre de procédure, un paramètre de type, un retour de fonction ou un tableau comme étant d'un type restreint.

Le Common Language Runtime (CLR) expose certains types uniquement pour la prise en charge de langage spéciale, et ils ne doivent pas être utilisés comme types de données dans votre application. Ces types sont les structures ArgIterator, RuntimeArgumentHandle et TypedReference.

ID d'erreur : BC31396

Pour corriger cette erreur

- N'utilisez pas la structure restreinte comme type de données déclaré.

Voir aussi

Référence

ArgIterator
RuntimeArgumentHandle
TypedReference

La version ciblée du .NET Compact Framework ne prend pas en charge l'utilisation du modificateur Ansi, Auto ou Unicode

The targeted version of the .NET Compact Framework does not support using the Ansi, Auto or Unicode modifier

La version du .NET Compact Framework que vous utilisez ne prend pas en charge les modificateurs **Ansi**, **Auto** et **Unicode**.

ID d'erreur : BC30763

Pour corriger cette erreur

- Supprimez le modificateur.

Voir aussi

Référence

Ansi
Auto
Unicode (Visual Basic)

La version ciblée du .NET Compact Framework ne prend pas en charge l'instruction 'End'

The targeted version of the .NET Compact Framework does not support the 'End' statement

La version du .NET Compact Framework que vous utilisez ne prend pas en charge l'instruction **End**.

ID d'erreur : BC30769

Pour corriger cette erreur

- Supprimez l'instruction.

Voir aussi

Référence

End (Visual Basic)

La version ciblée du .NET Compact Framework ne prend pas en charge la résolution de surcharge à liaison tardive

The targeted version of the .NET Compact Framework does not support latebound overload resolution

La version ciblée du .NET Compact Framework ne prend pas en charge la résolution de la surcharge à liaison tardive.

ID d'erreur : BC30764

Pour corriger cette erreur

- Effectuez le cast des paramètres d'objet de type en type approprié.

Voir aussi

Concepts

Résolution de surcharge

Autres ressources

Objets en Visual Basic

La version ciblée du .NET Compact Framework ne prend pas en charge la liaison tardive

The targeted version of the .NET Compact Framework does not support latebinding

La version du .NET Compact Framework que vous utilisez ne prend pas en charge la liaison tardive.

ID d'erreur : BC30762

Pour corriger cette erreur

1. Évitez d'appeler des fonctions, des subs ou des propriétés sur une variable déclarée comme un objet.
2. Évitez d'utiliser la variable objet comme un tableau.
3. Évitez d'utiliser les expressions d'accès au membre du dictionnaire avec des variables objet.

Voir aussi

Concepts

Caractères spéciaux dans le code

Autres ressources

Objets en Visual Basic

Le nombre maximal d'erreurs est dépassé

The maximum number of errors has been exceeded

Le nombre maximal d'erreurs du compilateur (100) a été atteint.

ID d'erreur : BC2020

Pour corriger cette erreur

- Réduisez le nombre d'erreurs générées en déboguant et en adressant chaque erreur tour à tour.

Voir aussi

Concepts

Types d'erreurs

Cet héritage provoque des dépendances circulaires entre <type1> '<NomType1>' et son <type2> '<NomType2>' imbriqué

This inheritance causes circular dependencies between <type1> '<typename1>' and its nested <type2> '<typename2>'

Une structure d'héritage entraîne une dépendance circulaire entre les classes imbriquées, c'est-à-dire entre deux classes qui héritent l'une de l'autre.

Le code suivant peut générer ce message d'erreur :

```
Public Class c1
    Inherits c3.c4
    Public Class c2
    End Class
End Class
Public Class c3
    Inherits c1.c2
    Public Class c4
    End Class
End Class
```

Dans le précédent code, la classe `c1` hérite de la classe `c4`, mais `c4` est imbriqué dans `c3`, qui hérite de `c2`, qui est imbriqué dans `c1`.

ID d'erreur : BC30907

Pour corriger cette erreur

- Modifiez la structure d'héritage afin qu'il n'existe aucune dépendance circulaire.

Voir aussi

Concepts

Éléments fondamentaux de l'héritage

Arguments trop nombreux pour '<nomprocédure>'

Too many arguments to '<procedurename>'

Le nombre d'arguments fourni dans un appel de procédure dépasse le nombre d'arguments défini pour la procédure.

ID d'erreur : BC30057

Pour corriger cette erreur

- Consultez la définition de la procédure et supprimez les arguments supplémentaires de l'appel.

Voir aussi

Concepts

Paramètres et arguments d'une procédure

Arguments de type insuffisants dans '<NomTypeGénérique>'

Too few type arguments to '<genericTypeName>'

Une classe, une structure, une interface ou un délégué générique est appelé avec moins d'arguments de type que de paramètres de type.

Lorsque vous appelez un type générique, vous devez fournir un seul argument de type pour chaque paramètre de type défini par ce type générique. Le compilateur n'essaie l'inférence du type que sur les procédures génériques et non sur les types génériques.

ID d'erreur : BC32042

Pour corriger cette erreur

- Ajoutez des arguments de type à votre liste d'arguments de type de sorte qu'il y en ait un pour chaque paramètre de type sur le type générique que vous appelez.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Les instructions 'Throw' ne sont pas valides dans la fenêtre Exécution

'Throw' statements are not valid in the Immediate window

Les instructions **Throw** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30725

Pour corriger cette erreur

- Supprimez les instructions **Throw** du code de débogage.

Voir aussi

Référence

Throw, instruction (Visual Basic)

Autres ressources

Débogage dans Visual Studio

L'instruction 'Throw' ne peut pas omettre l'opérande en dehors d'une instruction 'Catch' ou dans une instruction 'Finally'.

'Throw' statement cannot omit operand outside a 'Catch' statement or inside a 'Finally' statement

Les instructions **Throw** qui se trouvent à l'extérieur de l'instruction **Catch** doivent indiquer le nom d'un objet d'exception.

ID d'erreur : BC30666

Pour corriger cette erreur

1. Spécifiez le nom d'un objet d'exception dérivé de **System.Exception**.
2. Restructurez votre code de manière à ce que l'instruction **Throw** se trouve à l'intérieur d'un bloc **Catch**.

Voir aussi

Référence

Throw, instruction (Visual Basic)

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Classe d'exception dans Visual Basic

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

L'opérande 'Throw' doit dériver de 'System.Exception'

'Throw' operand must derive from 'System.Exception'

L'argument fourni pour **Throw** doit être une instance de **System.Exception** ou une instance d'une classe dérivée de **System.Exception**.

ID d'erreur : BC30665

Pour corriger cette erreur

- Utilisez un argument dérivé de **System.Exception**, comme dans l'exemple suivant.

```
Throw New System.Exception("This is an error.")
```

Voir aussi

Référence

Throw, instruction (Visual Basic)

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Classe d'exception dans Visual Basic

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

Un type dans un module ne peut pas être déclaré '<spécificateur>'

Type in a Module cannot be declared '<specifier>'

Un type a été déclaré dans un module avec des modificateurs non valides.

ID d'erreur : BC30735

Pour corriger cette erreur

- N'utilisez pas les modificateurs **Protected** ou **Protected Friend** sur les types déclarés dans un module.

Voir aussi

Référence

Module, instruction
Dim, instruction (Visual Basic)

Type attendu

Type expected

Les caractères de code source qui apparaissent ne peuvent pas être reconnus comme nom de type. Le contexte nécessite soit un type de données élémentaire, soit un nom d'élément déclaré représentant un type.

ID d'erreur : BC30182

Pour corriger cette erreur

- Vérifiez que le type a été défini.
- Vérifiez que le nom de type est correctement orthographié.

Voir aussi

Référence

Class, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Function, instruction (Visual Basic)
Property, instruction
Structure, instruction

Les caractères de déclaration de type ne sont pas valides dans ce contexte

Type declaration characters are not valid in this context

Vous avez tenté d'utiliser un caractère, tel que $, qui n'est pas valide dans ce contexte.

ID d'erreur : BC30468

Pour corriger cette erreur

- Supprimez le caractère.

Voir aussi

Concepts

Caractères de type

La contrainte de type ne peut pas être une classe 'NotInheritable'

Type constraint cannot be a 'NotInheritable' class

Une liste de contraintes inclut une classe qui est marquée comme NotInheritable.

Une liste de contraintes d'un paramètre de type ne peut accepter qu'une seule classe. Un argument de type fourni pour ce paramètre de type doit hériter de cette classe. Par conséquent, le paramètre de type ne peut pas accepter une classe *sealed*, ou **NotInheritable**, comme une contrainte.

ID d'erreur : BC32060

Pour corriger cette erreur

- Si le paramètre de type doit être en mesure d'hériter de la classe et que vous pouvez contrôler sa définition, supprimez la déclaration **NotInheritable** de la classe.
- Si la classe doit rester **NotInheritable**, vous ne pouvez pas l'utiliser comme une contrainte. Supprimez le nom de classe dans la liste de contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic

La contrainte de type '<expression>' n'est pas une classe ou une interface

Type constraint '<expression>' is not a class or interface

Une liste de contraintes contient une expression qui ne représente pas une contrainte valide sur un paramètre de type.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Vous pouvez spécifier les éléments requis suivants selon n'importe quelle combinaison :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit hériter d'une classe au plus
- L'argument de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder
- L'argument de type doit être un type référence ou un type valeur

ID d'erreur : BC32048

Pour corriger cette erreur

- Vérifiez que l'expression et ses éléments sont correctement orthographiés.
- Si l'expression ne répond pas à la précédente liste d'exigences, supprimez-la de la liste des contraintes.
- Si l'expression fait référence à une interface ou une classe, vérifiez que le compilateur a accès à cette interface ou classe. Vous devrez peut-être qualifier son nom et ajouter une référence à votre projet. Pour plus d'informations, consultez « Références aux projets» dans Résolution d'une référence lorsque plusieurs variables ont le même nom.

Voir aussi

Tâches

Comment : qualifier un nom d'élément déclaré

Comment : ajouter et supprimer des références dans Visual Studio (C#, J#)

Concepts

Types génériques en Visual Basic

Types valeur et types référence

Les caractères de type ne sont pas autorisés pour les alias Imports

Type characters are not allowed on Imports aliases

Un alias d'importation dans une instruction **Imports** contient au moins un caractère de type d'identificateur.

Un alias d'importation doit correspondre à un nom Visual Basic valide. Les caractères autorisés n'incluent pas les caractères de type d'identificateur (**%**, **&**, **@** **!**, **#** et **$**). Consultez Noms d'éléments déclarés.

ID d'erreur : BC31398

Pour corriger cette erreur

- Supprimez le ou les caractères de type d'identificateur de l'alias d'importation.

Voir aussi

Référence

Imports, instruction

Concepts

Caractères de type

Noms d'éléments déclarés

Résolution d'une référence lorsque plusieurs variables ont le même nom

Les caractères de type ne sont pas autorisés dans les identificateurs d'étiquette

Type characters are not allowed in label identifiers

Une étiquette d'instruction contient au moins un caractère de type d'identificateur.

Une étiquette d'instruction doit être un nom Visual Basic valide. Les caractères autorisés n'incluent pas les caractères de type d'identificateur (**%**, **&**, **@** **!**, **#** et **$**). Consultez Noms d'éléments déclarés.

ID d'erreur : BC31395

Pour corriger cette erreur

- Supprimez le ou les caractères de type d'identificateur de l'étiquette d'instruction.

Voir aussi

Tâches

Comment : étiqueter des instructions

Concepts

Caractères de type

Noms d'éléments déclarés

Le caractère de type ne peut pas être utilisé dans une déclaration de paramètre de type

Type character cannot be used in a type parameter declaration

Une déclaration de paramètre de type contient au moins un caractère de type d'identificateur.

Un paramètre de type d'un type générique doit être un nom Visual Basic valide. Les caractères autorisés n'incluent pas les caractères de type d'identificateur (**%**, **&**, **@** **!**, **#** et **$**). Consultez Noms d'éléments déclarés.

ID d'erreur : BC32041

Pour corriger cette erreur

- Supprimez le ou les caractères de type d'identificateur de la déclaration de paramètre de type.

Voir aussi

Référence

Liste de types

Concepts

Caractères de type

Noms d'éléments déclarés

Types génériques en Visual Basic

Un caractère de type ne peut pas être utilisé dans une déclaration 'Sub' car une 'Sub' ne retourne pas de valeur

Type character cannot be used in a 'Sub' declaration because a 'Sub' doesn't return a value

Une procédure **Sub**, à l'instar d'une procédure **Function**, est une procédure distincte qui peut prendre en charge des arguments et exécuter une série d'instructions. Contrairement à une procédure **Function**, une procédure **Sub** ne retourne pas de valeur et ne peut donc pas contenir une déclaration de type.

ID d'erreur : BC30303

Pour corriger cette erreur

- Remplacez la procédure **Sub** par une procédure **Function**.

Voir aussi

Concepts

Sub, procédures

Function, procédures

Le caractère de type '<nomcaractère>' ne correspond pas au type de données déclaré '<type>'

Type character '<charactername>' does not match declared data type '<type>'

Une variable est déclarée avec un seul type de données, mais il y est fait référence avec un caractère de type représentant un type de données incompatible, par exemple `K$ = 10` où `K` est déclaré **Integer**.

ID d'erreur : BC30277

Pour corriger cette erreur

- Modifiez le type de données déclaré de la variable ou bien changez ou supprimez le caractère de type dans la référence.

Voir aussi

Concepts

Caractères de type

Le caractère de type '<caractère>' ne peut pas être utilisé dans une déclaration avec un type explicite

Type character '<character>' cannot be used in a declaration with an explicit type

Vous avez utilisé un caractère de type, tel que **$**, lors de la spécification explicite d'un type.

ID d'erreur : BC30302

Pour corriger cette erreur

- Supprimez le caractère de type ou supprimez la clause `As <Type>` de la déclaration.

Voir aussi

Concepts

Caractères de type

Déclaration implicite et explicite

Arguments de type inattendus, car les attributs ne peuvent pas être génériques

Type arguments unexpected because attributes cannot be generics

Un attribut est appliqué à l'aide d'une liste d'arguments de type.

À l'heure actuelle, Visual Basic et .NET Framework ne prennent en charge aucune combinaison d'attributs et de types génériques. Cela signifie que les limitations suivantes s'appliquent :

- Un attribut ne peut pas être un type générique ou déclaré dans un type générique.
- Un attribut ne peut pas hériter d'une classe générique et vice versa.
- Lorsque vous appliquez un attribut, vous ne pouvez pas fournir un argument qui fait partie des types suivants :
    - un type générique ;
    - un type construit à partir d'un type générique ;
    - un paramètre de type d'un type conteneur ;
    - Un type construit à partir d'un paramètre de type d'un type conteneur.

ID d'erreur : BC32066

Pour corriger cette erreur

- Si les arguments de type sont conçus pour être des arguments normaux, supprimez le mot clé **Of**. Ainsi, la liste d'arguments de type est transformée en une liste d'arguments normaux.
- Si les arguments de type sont destinés aux paramètres de type, supprimez le mot clé **Of** et tous les arguments de type. Un attribut ne peut pas accepter des arguments de type.

Voir aussi

Référence

Liste de types

Attribute

Concepts

Vue d'ensemble des attributs dans Visual Basic

Types génériques en Visual Basic

Arguments de type inattendus

Type arguments unexpected

Une clause **Implements** fournit des arguments de type pour le membre d'interface qu'elle implémente.

La clause **Implements** doit identifier uniquement l'interface et le membre qu'elle implémente. En d'autres termes, si vous déclarez une procédure générique, la clause **Of** et les arguments de type doivent apparaître dans la partie principale de la déclaration, comme ce serait le cas si vous n'implémentiez pas une procédure d'interface.

Le code suivant peut générer ce message d'erreur.

```
Public Interface testInterface
    Sub testSub(Of t)()
End Interface
Public Class testClass
    Implements testInterface
    Public Sub testSub() Implements testInterface.testSub(Of t)()
    End Sub
End Class
```

La déclaration qui précède la clause **Implements** doit ressembler à la définition d'interface, sauf que des modificateurs d'accès ou de procédure peuvent être ajoutés. Le code suivant permet d'éviter le message d'erreur.

```
    Public Sub testSub(Of t)() Implements testInterface.testSub
```

ID d'erreur : BC32088

Pour corriger cette erreur

- Supprimez la liste des arguments de type de la clause **Implements**.
- Si vous implémentez un membre générique de l'interface, insérez la liste des arguments de type dans la partie principale de la déclaration, qui précède le mot clé **Implements**.

Voir aussi

Référence

Implements (Visual Basic)
Liste de types

Concepts

Implements, mot clé et instruction
Types génériques en Visual Basic
Procédures génériques dans Visual Basic

Les arguments de type inférés pour la méthode '<NomProcédure>' entraînent les avertissements suivants : <ListeAvertissements>

Type arguments inferred for method '<procedurename>' result in the following warnings :<warninglist>

Une procédure générique est appelée sans fournir d'arguments de type, et les arguments de type inférés produisent un ou plusieurs avertissements.

Normalement, lorsque vous appelez un type générique, vous fournissez un argument de type pour chaque paramètre de type que le type générique définit. Si vous ne fournissez pas d'arguments de type, le compilateur essaie d'inférer les types à passer aux paramètres de type. Si les types inférés entraînent une ambiguïté ou s'ils créent une situation qui pourrait donner des résultats inattendus, le compilateur génère cet avertissement.

Une *contrainte* sur un paramètre de type limite le type d'arguments qui peut lui être passé. Par exemple, un paramètre de type peut être contraint à être une classe qui implémente l'interface IComparable. Pour plus d'informations, consultez « Constraints » dans Procédures génériques dans Visual Basic.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC41006

Pour corriger cette erreur

- Fournissez des arguments de type à la procédure générique afin que le compilateur n'ait pas besoin de les inférer.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Les arguments de type inférés pour la méthode '<NomProcédure>' entraînent les erreurs suivantes : <ListeErreurs>

Type arguments inferred for method '<procedurename>' result in the following errors :<errorlist>

Une procédure générique est appelée sans fournir d'arguments de type, et les arguments de type inférés provoquent une ou plusieurs violations de contrainte.

En général, lorsque vous appelez un type générique, vous fournissez un argument de type pour chaque paramètre de type défini par le type générique. Si vous ne fournissez pas d'arguments de type, le compilateur essaie de déduire les types à passer aux paramètres de type. Si les types inférés ne satisfont pas une ou plusieurs des contraintes de paramètre de type, le compilateur génère cette erreur.

Une *contrainte* sur un paramètre de type limite les arguments de type qui peuvent lui être passés. Par exemple, un paramètre de type peut être contraint à être une classe qui implémente l'interface IComparable. Pour plus d'informations, consultez « Constraints » dans Procédures génériques dans Visual Basic.

ID d'erreur : BC30954

Pour corriger cette erreur

- Fournissez des arguments de type à la procédure générique, afin que le compilateur ne les infère pas.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Les arguments de type pour la méthode '<NomProcédure>' n'ont pas pu être inférés à partir du délégué '<NomDélégué>'

Type arguments for method '<procedurename>' could not be inferred from the delegate '<delegatename>'

Une instruction d'assignation utilise **AddressOf** pour assigner l'adresse d'une procédure générique à un délégué, mais elle ne fournit pas d'arguments de type à la procédure générique.

En général, lorsque vous appelez un type générique, vous fournissez un argument de type pour chaque paramètre de type défini par le type générique. Si vous ne fournissez pas d'arguments de type, le compilateur essaie d'inférer les types à passer aux paramètres de type. Si le contexte ne fournit pas assez d'informations pour que le compilateur puisse inférer les types, il génère une erreur.

ID d'erreur : BC30952

Pour corriger cette erreur

- Spécifiez les arguments de type de la procédure générique dans l'expression **AddressOf**.

Voir aussi

Référence

AddressOf, opérateur

Liste de types

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Les arguments de type ne peuvent pas être appliqués à l'expression '<expression>'

Type arguments cannot be applied to the expression '<expression>'

Un alias d'importation est défini avec une clause Of qui lui passe les arguments de type.

Les arguments de type sont utilisés pour les types génériques et seuls les classes, les structures, les interfaces, les procédures et les délégués peuvent être génériques. Les espaces de noms et les alias d'importation ne peuvent pas être génériques.

ID d'erreur : BC32058

Pour corriger cette erreur

- Supprimez la clause **Of** et ses arguments de type de l'alias d'importation.

Voir aussi

Référence

Imports, instruction

Liste de types

Concepts

Références et l'instruction Imports

Résolution d'une référence lorsque plusieurs variables ont le même nom

Types génériques en Visual Basic

L'inférence d'argument de type a échoué pour le paramètre de type '<NomParamètreDeType>' de '<SignatureDeProcédureGénérique>'

Message d'erreur
L'inférence d'argument de type a échoué pour le paramètre de type '<NomParamètreDeType>' de '<SignatureDeProcédureGénérique>'. L'argument de type n'a pas pu être inféré à partir de l'argument passé au paramètre '<NomParamètre>'.
Type argument inference failed for type parameter '<typeparametername>' of '<genericproceduresignature>'. Type argument could not be inferred from the argument passed to parameter '<parametername>'.

Une procédure générique est appelée sans fournir d'arguments de type, et le compilateur ne peut pas déduire le type à passer à l'un des paramètres.

En général, lorsque vous appelez une procédure générique, vous fournissez un argument de type pour chaque paramètre de type défini par la procédure générique. Si vous ne fournissez pas d'arguments de type, le compilateur essaie de déduire les types à passer aux paramètres de type. Si le contexte de l'appel fournit des informations de type de données incompatibles pour un paramètre de type, l'inférence de type échoue.

Le code suivant peut générer ce message d'erreur.

```
Public Sub doSomething(Of t)(ByVal arg1 As t(), ByVal arg2 As t)
End Sub
Call doSomething(6, 42)
```

Dans l'exemple précédent, le compilateur infère le type **Integer** pour `t` en fonction de la valeur 42 passée à `arg2`. Toutefois, cette inférence requiert que `arg1` soit de type **Integer()**, c'est-à-dire un tableau de **Integer**, et la valeur 6 passée à `arg1` ne correspond pas à ce type.

ID d'erreur : BC32051

Pour corriger cette erreur

- Fournissez des arguments de type à la procédure générique, afin que le compilateur ne les infère pas.
- Fournissez des arguments normaux dont les types correspondent à ceux des arguments de type.

Voir aussi

Référence
Liste de types
Concepts
Types génériques en Visual Basic
Procédures génériques dans Visual Basic

L'inférence d'argument de type a échoué pour le paramètre de type '<NomParamètreDeType1>' de '<SignatureDeProcédureGénérique>'

Message d'erreur
L'inférence d'argument de type a échoué pour le paramètre de type '<NomParamètreDeType1>' de '<SignatureDeProcédureGénérique>'. L'argument de type inféré de l'argument passé au paramètre' <NomParamètre1>' est en conflit avec l'argument de type inféré de l'argument passé au paramètre' <NomParamètre2>.
Type argument inference failed for type parameter '<typeparametername1>' of '<genericproceduresignature>'. Type argument inferred from the argument passed to parameter '<parametername1>' conflicts with the type argument inferred from the argument passed to parameter '<parametername2>'.

Une procédure générique est appelée sans argument de type, et l'inférence de type tentée a généré un conflit de type de données entre les paramètres de type.

En général, lorsque vous appelez une procédure générique, vous fournissez un argument de type pour chaque paramètre de type défini par la procédure générique. Si vous ne fournissez pas d'arguments de type, le compilateur essaie de déduire les types à passer aux paramètres de type. Si le contexte de l'appel fournit des informations de type de données incompatibles pour un paramètre de type, l'inférence de type échoue.

Le code suivant peut générer ce message d'erreur.

```
Public Sub takeTwoValues(Of t)(ByVal x As t, ByVal y As t)
End Sub
Call takeTwoValues(4, 2.5)
```

Étant donné que le premier argument provoque l'inférence de **Integer** par le compilateur pour le paramètre de type `t`, et que le deuxième argument provoque l'inférence de **Double** par le compilateur pour le même paramètre de type, il existe un conflit sur le type de données à passer à `t`.

ID d'erreur : BC32116

Pour corriger cette erreur

- Fournissez des arguments de type au type générique, afin que le compilateur ne les infère pas.

  ```
  Call takeTwoValues(Of Double)(4, 2.5)
  ```

  Notez que dans ce cas, où les deux arguments normaux sont de types de données différents, le code appelant doit passer un argument de type qui peut prendre en charge ces deux types de données. Dans ce cas, **Integer** s'étend à **Double**.

  - ou -

- Redéfinissez la procédure générique pour spécifier des paramètres de type différents pour les paramètres normaux, afin qu'il existe aucun conflit inférant les types.

  ```
  Public Sub takeTwoValues(Of t1, t2)(ByVal x As t1, ByVal y As t2)
  ```

Voir aussi

Référence
Liste de types
Concepts
Types génériques en Visual Basic
Procédures génériques dans Visual Basic

L'argument de type '<NomArgumentDeType>' doit avoir un constructeur d'instance sans paramètre public pour satisfaire la contrainte 'New' pour le paramètre de type '<NomParamètreDeType>'

Type argument '<typeargumentname>' must have a public parameterless instance constructor to satisfy the 'New' constraint for type parameter '<typeparametername>'

Un argument de type fournit un type sans un constructeur sans paramètre accessible à un paramètre de type avec la contrainte New (Visual Basic).

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Une exigence possible est que l'argument de type doit exposer un constructeur sans paramètre auquel le code de création peut accéder. Pour spécifier cette exigence, la liste des contraintes contient la contrainte **New**.

ID d'erreur : BC32083

Pour corriger cette erreur

1. Vérifiez que le nom de type générique et le nom de type dans l'argument de type sont correctement orthographiés.
2. Choisissez un type pour l'argument de type qui expose un constructeur sans paramètre accessible. Vous ne pouvez pas appeler ce type générique particulier sauf si vous pouvez fournir un tel argument de type à ce paramètre de type.

Voir aussi

Tâches

Comment : utiliser une classe générique

Référence

Liste de types

Concepts

Types génériques en Visual Basic

L'argument de type '<NomArgumentDeType>' ne satisfait pas la contrainte 'Structure' pour le paramètre de type '<NomParamètreDeType>'

Type argument '<typeargumentname>' does not satisfy the 'Structure' constraint for type parameter '<typeparametername>'

Un argument de type fourni à un type générique ne satisfait pas la contrainte de type valeur sur son paramètre de type correspondant.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Si la liste des contraintes ne contient pas de classes ou d'interfaces spécifiques, vous pouvez imposer une exigence générale en spécifiant l'un des éléments suivants :

- L'argument de type doit être un type valeur (inclure la contrainte Structure (Visual Basic))
- L'argument de type doit être un type référence (inclure la contrainte Class (Visual Basic))

Vous ne pouvez pas spécifier **Structure** et **Class** pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32105

Pour corriger cette erreur

- Sélectionnez un argument de type d'un type valeur quelconque.

Voir aussi

Tâches

Comment : utiliser une classe générique

Concepts

Types génériques en Visual Basic

Types valeur et types référence

L'argument de type '<NomArgumentDeType>' ne satisfait pas la contrainte 'Class' pour le paramètre de type '<NomParamètreDeType>'

Type argument '<typeargumentname>' does not satisfy the 'Class' constraint for type parameter '<typeparametername>'

Un argument de type fourni à un type générique ne satisfait pas la contrainte de type référence sur son paramètre de type correspondant.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Si la liste des contraintes ne contient pas de classes ou d'interfaces spécifiques, vous pouvez imposer une exigence générale en spécifiant l'un des éléments suivants :

- L'argument de type doit être un type valeur (inclure la contrainte Structure (Visual Basic))
- L'argument de type doit être un type référence (inclure la contrainte Class (Visual Basic))

Vous ne pouvez pas spécifier **Structure** et **Class** pour le même paramètre de type, de même que vous ne pouvez pas spécifier l'un ou l'autre plusieurs fois.

ID d'erreur : BC32106

Pour corriger cette erreur

- Sélectionnez un argument de type d'un type référence quelconque.

Voir aussi

Tâches

Comment : utiliser une classe générique

Concepts

Types génériques en Visual Basic

Types valeur et types référence

L'argument de type '<NomArgumentDeType>' n'hérite pas ou n'implémente pas le type de contrainte '<NomParamètreDeType>'

Type argument '<typeargumentname>' does not inherit from or implement the constraint type '<typeparametername>'

Un argument de type fourni à un type générique ne satisfait pas la contrainte d'héritage ou d'implémentation de son paramètre de type correspondant.

Une liste de contraintes impose des exigences sur l'argument de type passé au paramètre de type. Les exigences possibles sont les suivantes :

- L'argument de type doit implémenter une ou plusieurs interfaces
- L'argument de type doit hériter d'une classe au plus

Vous pouvez combiner les exigences précédentes pour un seul paramètre de type. Visual Basic ne peut pas construire le type à moins que le code ne fournisse des arguments de type qui satisfont chaque contrainte sur chaque paramètre de type affecté au type générique.

ID d'erreur : BC32044

Pour corriger cette erreur

- Sélectionnez un argument de type d'un type qui implémente chaque interface spécifiée pour le paramètre de type et qui hérite de la classe spécifiée s'il en existe un.

Voir aussi

Tâches

Comment : utiliser une classe générique

Concepts

Types génériques en Visual Basic

L'argument de type '<NomArgumentDeType>' est déclaré 'MustInherit' et ne satisfait pas la contrainte 'New' pour le paramètre de type '<NomParamètreDeType>'

Type argument '<typeargumentname>' is declared 'MustInherit' and does not satisfy the 'New' constraint for type parameter '<typeparametername>'

Un type générique est appelé avec la classe **MustInherit** comme argument de type, alors que le paramètre de type correspondant est déclaré avec la contrainte **New**.

La contrainte **New** requiert que le type passé dans l'argument de type correspondant prenne en charge la création d'objets. Toutefois, une classe *abstraite*, c'est-à-dire une classe déclarée comme **MustInherit**, n'expose pas de constructeurs car vous ne pouvez pas créer d'objets à partir de celle-ci.

ID d'erreur : BC32082

Pour corriger cette erreur

1. Si la classe utilisée dans l'argument de type ne doit pas être abstraite, supprimez le mot clé **MustInherit** de sa déclaration.
2. Si la classe utilisée dans l'argument de type doit être abstraite, mais ne doit pas être utilisée pour construire le type générique, passez une autre classe dans l'argument de type.
3. Si le paramètre de type correspondant ne doit pas créer d'objets du type qui lui est passé, supprimez la contrainte **New** de sa déclaration.

Voir aussi

Référence

New (Visual Basic)

MustInherit

Concepts

Types génériques en Visual Basic

Le type '<NomType1>' ne peut pas être marqué comme conforme CLS, car son type conteneur '<NomType2>' n'est pas conforme CLS

Type '<typename1>' cannot be marked CLS-compliant because its containing type '<typename2>' is not CLS-compliant

Une classe ou une interface est marquée comme `<CLSCompliant(True)>` lorsqu'elle est imbriquée dans un type qui est marqué comme `<CLSCompliant(False)>` ou qui n'est pas marqué.

Pour qu'une classe ou une interface soit conforme au Spécification CLS (Common Language Specification) (CLS), toute sa hiérarchie de relation contenant-contenu doit être conforme. Cela signifie que chaque type dans lequel elle est imbriquée doit être conforme.

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40030

Pour corriger cette erreur

- S'il est nécessaire d'avoir la conformité CLS, définissez ce type dans une hiérarchie de relation contenant-contenu différente.
- S'il doit rester dans sa hiérarchie de relation contenant-contenu actuelle, supprimez l'**CLSCompliantAttribute** de sa définition ou marquez-le comme `<CLSCompliant(False)>`.

Voir aussi

Concepts

Écriture d'un code conforme CLS

Le type '<NomType>' doit définir l'opérateur '<opérateur>' à utiliser dans une instruction 'For'

Type '<typename>' must define operator '<operator>' to be used in a 'For' statement

Une boucle **For** spécifie une variable compteur d'un type qui ne prend pas en charge un opérateur requis.

La variable compteur dans une boucle **For** peut être de n'importe quel type de données qui prend en charge tous les opérateurs suivants :

- supérieur ou égal (**>=**) ;
- inférieur ou égal (**<=**) ;
- addition (**+**) ;
- soustraction (**-**).

Si vous utilisez un type de données numériques pour la variable compteur, tous les opérateurs précédents sont pris en charge. Si vous utilisez une classe ou une structure définie par l'utilisateur, vous devez définir tous les opérateurs précédents sur cette classe ou structure.

Notez également que les types de données des expressions *start*, *end* et *step* dans l'instruction **For** doivent s'étendre au type de données de la variable compteur. Si la variable compteur est une classe ou une structure définie par l'utilisateur et si l'expression *start*, *end* ou *step* est d'un type différent, vous devez définir l'opérateur de conversion **CType** pour exécuter la conversion nécessaire.

ID d'erreur : BC33038

Pour corriger cette erreur

1. Assurez-vous que l'orthographe du type de données de la variable de compteur est correcte.
2. Si vous utilisez une classe ou une structure définie par l'utilisateur pour la variable compteur, définissez tous les opérateurs requis sur cette classe ou structure.
3. Selon les types de données des expressions *start*, *end* et *step*, vous devrez peut-être définir un ou plusieurs opérateurs de conversion **CType** pour les convertir vers le type de données de la variable compteur.

Voir aussi

Tâches

Comment : définir un opérateur
Comment : définir un opérateur de conversion

Référence

For...Next, instruction (Visual Basic)
Operator, instruction
CType, fonction

Concepts

Procédures d'opérateur

Le type '<NomType>' doit définir l'opérateur '<OpérateurDéterminant>' à utiliser dans une expression '<OpérateurCourtCircuit>'

Type '<typename>' must define operator '<determinantoperator>' to be used in a '<shortcircuitoperator>' expression

Un opérateur AndAlso ou OrElse Operator utilise des opérandes d'un type de classe ou de structure, lorsque cette classe ou structure ne définit pas d'opérateur requis.

Étant donné que vous ne définissez pas directement un opérateur de court-circuit (**AndAlso** ou **OrElse**), vous devez définir les opérateurs logiques et déterminants correspondants. Le tableau suivant présente les opérateurs requis.

Opérateur de court-circuit	Opérateur logique	Opérateur déterminant
AndAlso	And, opérateur (Visual Basic)	IsFalse, opérateur
OrElse	Or, opérateur (Visual Basic)	IsTrue, opérateur

Visual Basic utilise ces opérateurs logiques et déterminants pour construire la logique « court-circuitante » pour **AndAlso** ou **OrElse**. Pour que cela fonctionne correctement, les opérandes et la valeur de retour de votre définition **And** ou **Or** doivent être du type conteneur, c'est-à-dire du type de la classe ou la structure dans laquelle vous définissez **And** ou **Or**.

ID d'erreur : BC33035

Pour corriger cette erreur

- Définissez les opérateurs **And** et **IsFalse**, ou les opérateurs **Or** et **IsTrue**, dans la classe ou la structure utilisée pour le type d'opérande de l'opérateur **AndAlso** ou **OrElse**. Vérifiez que les opérandes pour **And** ou **Or** sont du type de la classe ou de la structure dans laquelle vous le définissez.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Opérateurs de bits et opérateurs logiques en Visual Basic

Type '<nomtype>' non défini

Type '<typename>' is not defined

L'instruction a fait référence à un type qui n'a pas été défini. Vous pouvez définir un type dans une instruction de déclaration, telle que **Enum**, **Structure**, **Class** ou **Interface**.

ID d'erreur : BC30002

Pour corriger cette erreur

1. Vérifiez que la définition de type et sa référence utilisent toutes les deux la même orthographe.
2. Vérifiez que la définition de type est accessible à la référence. Par exemple, si le type se trouve dans un autre module et a été déclaré **Private**, déplacez la définition de type vers le module de référence ou déclarez-le **Public**.
3. Vérifiez que l'espace de noms du type n'est pas redéfini au sein de votre projet. Si tel est le cas, utilisez le mot clé **Global** pour qualifier intégralement le nom du type. Par exemple, si un projet définit un espace de noms nommé `System`, il n'est pas possible d'accéder au type System.Object à moins que celui-ci ne soit entièrement qualifié avec le mot clé **Global** : `Global.System.Object`.
4. Si le type est défini, mais que la bibliothèque d'objets ou de types dans laquelle il est défini n'est pas inscrite dans Visual Basic, cliquez sur **Ajouter une référence** dans le menu **Projet**, puis sélectionnez la bibliothèque d'objets ou de types appropriée.

Voir aussi

Référence

Enum, instruction (Visual Basic)
Structure, instruction
Class, instruction (Visual Basic)
Interface, instruction (Visual Basic)
Global
Ajouter une référence, boîte de dialogue

Concepts

Espaces de noms dans Visual Basic

Le type '<nomtype>' n'est pas défini ou le module le contenant n'est pas chargé dans la session de débogage

Type '<typename>' is not defined or the module containing it is not loaded in the debugging session

Un type non disponible ou non défini a été référencé.

ID d'erreur : BC30743

Pour corriger cette erreur

1. Vérifiez que le nom du type est correctement orthographié.
2. Définissez le type dans la classe ou le module en cours.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Le type <NomType> n'est pas conforme CLS

Type <typename> is not CLS-compliant

Une variable, une propriété ou un retour de fonction est déclaré avec un type de données qui n'est pas conforme CLS.

Pour qu'une application soit conforme au Spécification CLS (Common Language Specification) (CLS), elle doit utiliser uniquement des types conformes CLS.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

ID d'erreur : BC40041

Pour corriger cette erreur

- Si votre application doit être conforme CLS, remplacez le type de données de cet élément par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.
- Si votre application ne doit pas être conforme CLS, aucune modification n'est nécessaire. Toutefois, vous devez être conscient de sa non-conformité.

Voir aussi

Concepts

Écriture d'un code conforme CLS

Le type '<nomtype>' est importé à partir de différentes versions de l'assembly '<nomassembly>'

Message d'erreur
Le type '<nomtype>' est importé à partir de différentes versions de l'assembly '<nomassembly>' Il est impossible d'utiliser des versions différentes du même assembly avec nom fort dans le même projet.
Type '<typename>' is imported from different versions of assembly '<assemblyname>'. Different versions of the same assembly cannot be used in the same project.

Un type a été importé à partir de différentes versions du même assembly, ce qui a créé une ambiguïté.

ID d'erreur : BC31525

Pour corriger cette erreur

- Supprimez les références aux différentes versions du même assembly.

Voir aussi

Concepts
Références de projet

Le type '<NomType>' n'a pas de paramètres de type et donc ne peut pas avoir d'arguments de type

Type '<typename>' has no type parameters and so cannot have type arguments

Une déclaration ou une instruction d'assignation inclut une clause Of lors de l'appel d'un type non générique.

Par définition, un *type générique* est une classe, une structure, une interface, une procédure ou un délégué qui fonctionne sur des types de données que vous pouvez spécifier dans un ou plusieurs *paramètres de type*. Lorsque le code d'utilisation crée un type à partir de ce type générique, il fournit un *argument de type* à chaque paramètre de type. Lors de la création du type, chaque argument de type remplace toutes les occurrences de son paramètre de type correspondant dans le code généré.

Les paramètres de type sont définis avec une clause **Of** à l'intérieur des parenthèses et les arguments de type sont fournis à l'aide d'une clause **Of** à l'intérieur des parenthèses. La clause **Of** est utilisée uniquement lors de la gestion des types génériques.

Les types non génériques n'acceptent pas de paramètres de type et vous ne pouvez pas spécifier des arguments de type lorsque vous appelez un tel type.

ID d'erreur : BC32045

Pour corriger cette erreur

1. Vérifiez l'orthographe du type que vous utilisez dans la déclaration ou l'instruction d'assignation.
2. Si vous appelez un type non générique, supprimez la clause **Of** et ses parenthèses, le cas échéant. Ne supprimez pas les parenthèses qui entourent une liste d'arguments standard pour une procédure, un délégué ou un constructeur de classes.

Voir aussi

Tâches

Comment : utiliser une classe générique

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Le type '<nomtype>' n'a aucun constructeur

Type '<typename>' has no constructors

Un type ne prend pas en charge un appel à **Sub New()**. L'une des causes possibles est un compilateur ou un fichier binaire endommagé.

ID d'erreur : BC30251

Pour corriger cette erreur

1. Si le type se trouve dans un projet différent ou dans un fichier référencé, réinstallez le projet ou le fichier.
2. Si le type se trouve dans le même projet, recompilez l'assembly contenant le type.
3. Si l'erreur se reproduit, réinstallez le compilateur Visual Basic.
4. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

Autres ressources

Support technique et accessibilité

Le type '<NomType>' ne peut pas hériter d'un paramètre de type

Type '<typename>' cannot inherit from a type parameter

Une classe ou une interface contient une instruction Inherits qui spécifie un paramètre de type générique.

Un type ne peut pas hériter d'un type qui n'est pas encore défini. L'héritage implique la possibilité de réutiliser des membres de la classe de base, qui requiert à son tour que ces membres soient définis. Un paramètre de type générique est un espace réservé qui doit être remplacé par un type spécifique fourni par un argument de type. Par conséquent, un type ne peut pas hériter de l'espace réservé.

ID d'erreur : BC32055

Pour corriger cette erreur

- Si le type d'héritage doit hériter d'un autre type, utilisez un type spécifique au lieu d'un paramètre de type.
- Si le type de base doit être représenté par un paramètre de type générique, aucun autre type ne peut en hériter. Supprimez l'instruction Inherits.

Voir aussi

Concepts

Types génériques en Visual Basic

Autres ressources

Héritage en Visual Basic

Le type '<NomType>' ne peut pas hériter d'un type imbriqué dans celui-ci

Type '<typename>' cannot inherit from a type nested within it

Une classe ou une définition d'interface inclut une instruction Inherits qui spécifie un type imbriqué dans celui-ci.

L'héritage doit être linéaire et non circulaire. Un type ne peut pas hériter d'un autre type qui hérite de celui-ci.

Une restriction connexe signifie qu'un type ne peut pas hériter d'un type qui n'est pas encore défini. L'héritage implique la possibilité de réutiliser les membres de la classe de base, ce qui nécessite également la définition de ces membres. Par conséquent, Visual Basic ne peut pas compiler l'exemple de code suivant.

```
Public Class outerClass
    ' The following statement is INVALID because innerClass is not defined.
    Inherits innerClass
    Public Class innerClass
        Public Sub doSomething()
        End Sub
    End Class
End Class
```

ID d'erreur : BC30908

Pour corriger cette erreur

- Si le type héritant (type externe de l'imbrication) doit hériter du type interne, déplacez le type interne hors du type externe.
- Si le type interne doit être imbriqué dans le type externe, le type externe ne peut pas hériter de celui-ci. Supprimez l'instruction Inherits.

Voir aussi

Autres ressources

Héritage en Visual Basic

Impossible d'utiliser le type '<nomtype>' dans un attribut, car son conteneur '<nomconteneur>' n'est pas déclaré 'Public'

Type '<typename>' cannot be used as an attribute because its container '<containername>' is not declared 'Public'

La classe ou le module dans lequel cet attribut est défini n'est pas déclaré à l'aide du modificateur **Public**. Les classes et les modules qui ne spécifient aucun modificateur d'accès sont déclarés en tant que **Friend** par défaut.

ID d'erreur : BC31517

Pour corriger cette erreur

- Ajoutez le modificateur **Public** à la classe ou au module dans lequel cet attribut est défini.

Voir aussi

Référence

Public (Visual Basic)

Impossible d'utiliser le type '<nomtype>' dans un attribut, car il n'est pas déclaré 'Public'

Type '<typename>' cannot be used as an attribute because it is not declared 'Public'

Les attributs doivent être publics.

ID d'erreur : BC31516

Pour corriger cette erreur

- Ajoutez le modificateur **Public** à la classe d'attributs.

Voir aussi

Référence

Public (Visual Basic)

Le type '<NomType>' et le type partiel '<NomType>' déclarés dans '<NomFichier>' sont en conflit dans le conteneur '<NomConteneur>', mais sont fusionnés, car l'un d'eux est déclaré partiel

Type '<typename>' and partial type '<typename>' declared in '<filename>' conflict in container '<containername>', but are being merged because one of them is declared partial

Une classe ou une structure apparaît dans plusieurs définitions du même type de conteneur, et plusieurs définitions ne sont pas marquées **Partial**.

Vous devez utiliser le mot clé Partial (Visual Basic) sur au moins une des définitions d'une classe ou d'une structure, mais il est recommandé de l'utiliser sur toutes les définitions partielles.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40047

Pour corriger cette erreur

- Utilisez le mot clé Partial (Visual Basic) sur chaque définition partielle de la classe ou la structure.

Type '<NomType>' et type '<NomType>' partiel sont en conflit dans '<NomConteneur>', mais sont fusionnés, car l'un d'eux est déclaré partiel

Type '<typename>' and partial type '<typename>' conflict in container '<containername>', but are being merged because one of them is declared partial

Une classe ou une structure apparaît dans de nombreuses définitions du même type conteneur et plusieurs définitions ne sont pas marquées comme **Partial**.

Vous devez utiliser le mot clé Partial (Visual Basic) au moins sur l'une des nombreuses définitions d'une classe ou d'une structure, mais il est recommandé de l'utiliser sur toutes les définitions partielles.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40046

Pour corriger cette erreur

- Utilisez le mot clé Partial (Visual Basic) sur chaque définition partielle de la classe ou de la structure.

Voir aussi

Référence

Partial (Visual Basic)

Le type '<NomType>' n'est pas pris en charge, car il hérite directement ou indirectement de lui-même

Type '<typename>' is not supported because it either directly or indirectly inherits from itself

Une classe ou une interface hérite d'elle-même ou d'une autre classe ou interface qui, au final, hérite de celle-ci.

Visual Basic ne prend pas en charge l'héritage circulaire.

ID d'erreur : BC30916

Pour corriger cette erreur

- Modifiez la structure d'héritage afin qu'elle repose sur une interface ou classe de base qui n'hérite d'aucune autre classe ou interface.

Voir aussi

Concepts

Éléments fondamentaux de l'héritage

Le type '<NomType>' dans l'assembly '<NomAssembly1>' a été transmis à l'assembly '<NomAssembly2>'

Message d'erreur
Le type '<NomType>' dans l'assembly '<NomAssembly1>' a été transmis à l'assembly '<NomAssembly2>'. Soit une référence à '<NomAssembly2>' est absente du projet soit le type '<NomType>' est absent de l'assembly '<NomAssembly2>'. Type '<typename>' in assembly '<assemblyname1>' has been forwarded to assembly '<assemblyname2>'. Either a reference to '<assemblyname2>' is missing from your project or the type '<typename>' is missing from assembly '<assemblyname2>'.

Une expression du code source d'un assembly fait référence à un type qui a été transmis à un autre assembly, mais ce type est introuvable dans l'assembly de destination.

Le *Transfert de type* consiste à réassigner la définition d'une classe, d'une structure, d'une interface, d'un délégué ou d'une énumération à un assembly autre que celui dans lequel il a été initialement défini. Il est souvent utilisé conjointement avec la *Refactorisation de code* qui consiste à fractionner un assembly en deux ou plusieurs assemblys ou à déplacer le code d'un assembly à un autre.

Bien que le type soit toujours disponible temporairement dans l'assembly d'origine, il ne sera probablement plus défini après sa suppression de l'assembly d'origine par la refactorisation de code.

ID d'erreur : BC31424

Pour corriger cette erreur

- Assurez-vous que le type est présent dans l'assembly de destination.
- Assurez-vous que votre projet contient une référence à l'assembly de destination.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Référence
Type Forwarding
TypeForwardedToAttribute
Concepts
Références de projet

L'opérande 'TryCast' doit être un paramètre de type contraint par classe, mais '<NomParamètreDeType>' n'a pas de contrainte de classe

'TryCast' operands must be class-constrained type parameters, but '<typeparametername>' has no class constraint

L'opérateur TryCast est utilisé avec un opérande de paramètre de type qui n'est pas forcément un type référence.

TryCast utilise uniquement des types référence, tels que les classes ou les interfaces. Lorsque vous passez un paramètre de type en tant qu'argument à **TryCast**, vous devez contraindre ce paramètre de type à être un type référence. Pour ce faire, vous devez inclure un ou plusieurs des éléments suivants dans la liste de contraintes du paramètre de type :

- un ou plusieurs noms d'interfaces (l'argument de type doit tous les implémenter) ;
- au plus un nom de classe (l'argument de type doit en hériter) ;
- la contrainte New (Visual Basic) (l'argument de type devant exposer un constructeur sans paramètre auquel le code de création peut accéder, celui-ci doit être une classe) ;
- la contrainte Class (Visual Basic) (l'argument de type doit être un type référence).

ID d'erreur : BC30793

Pour corriger cette erreur

- Si vous devez passer ce paramètre de type à **TryCast**, imposez-lui une contrainte avec une ou plusieurs des contraintes de la liste précédente.
- Si le paramètre de type ne peut pas accepter uniquement un type référence, vous ne pouvez pas l'utiliser avec **TryCast**. Vous pouvez utiliser la fonction CType à la place.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic
Types valeur et types référence
Conversions étendues et restrictives
Conversions implicites et explicites

L'opérande 'TryCast' doit être un type référence, mais '<NomType>' est un type valeur

'TryCast' operand must be reference type, but '<typename>' is a value type

L'opérateur **TryCast** est utilisé avec un type valeur pour au moins l'un des arguments.

TryCast teste une relation d'héritage ou d'implémentation entre les deux arguments. Par conséquent, seuls les types référence sont autorisés pour les arguments. Pour plus d'informations, consultez Types valeur et types référence.

ID d'erreur : BC30792

Pour corriger cette erreur

- Utilisez **DirectCast** ou **CType** pour effectuer la conversion. Tous deux acceptent les types valeur.

Voir aussi

Référence

TryCast
DirectCast
CType, fonction

Concepts

Types valeur et types référence

Les instructions 'Try' ne sont pas valides dans la fenêtre Exécution

'Try' statements are not valid in the Immediate window

Les instructions **Try** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30717

Pour corriger cette erreur

- Supprimez les instructions **Try** du code de débogage.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Débogage dans Visual Studio

Try doit utiliser au moins un 'Catch' ou 'Finally'

Try must have at least one 'Catch' or a 'Finally'

Un bloc **Try** doit inclure un bloc **Finally** ou au moins un bloc **Catch** avant l'instruction **End Try**

ID d'erreur : BC30030

Pour corriger cette erreur

- Ajoutez un bloc **Catch** ou **Finally** dans le bloc **Try**.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

'Try' doit se terminer par un 'End Try' correspondant

'Try' must end with a matching 'End Try'

Try sert à initialiser un bloc **Try** ; elle ne peut dès lors apparaître qu'au début du bloc, avec une instruction **End Try** correspondante qui termine le bloc. Vous possédez une instruction **Try** redondante ou vous n'avez pas terminé votre bloc **Try** par **Finally**.

ID d'erreur : BC30384

Pour corriger cette erreur

- Recherchez et supprimez l'instruction **Try** superflue ou terminez le bloc par une instruction **End Try** correspondante.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Vue d'ensemble de la gestion structurée des exceptions pour Visual Basic

Arguments de type trop nombreux dans '<NomTypeGénérique>'

Too many type arguments to '<genericTypeName>'

Une classe, une structure, une interface ou un délégué générique est appelé avec plus d'arguments de type que de paramètres de type.

Lorsque vous appelez un type générique, vous devez fournir un seul argument de type pour chaque paramètre de type défini par ce type générique.

ID d'erreur : BC32043

Pour corriger cette erreur

- Supprimez les arguments de type de votre liste d'arguments de type de sorte qu'il en reste un seul pour chaque paramètre de type sur le type générique que vous appelez.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Impossible de générer une référence au fichier '<nomfichier>' (utilisez l'utilitaire TLBIMP pour faire référence aux DLL COM) : <message d'erreur>

Unable to generate a reference to file '<filename>' (use TLBIMP utility to reference COM DLLs): <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la recherche ou de la validation d'un fichier DLL COM+.

ID d'erreur : BC30142

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si la référence souhaitée porte sur une DLL COM au lieu d'une DLL COM+, utilisez l'outil Type Library Importer (Tlbimp.exe) pour générer la référence.
3. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)
Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)
Type Library Importer (Tlbimp.exe)

Autres ressources

Support technique et accessibilité

Le fichier requis '<nomfichier>' est introuvable

Unable to find required file '<filename>'

Un fichier requis par Microsoft Visual Studio 2005 est manquant ou endommagé.

ID d'erreur : BC30655

Pour corriger cette erreur

- Réinstallez Microsoft Visual Studio 2005.

Voir aussi

Autres ressources

Support technique

Impossible de trouver le point d'entrée '<nom>' dans la DLL '<nomfichier>' : <erreur>

Unable to find entry point '<name>' in DLL '<filename>': <error>

Le nom de la procédure est introuvable dans la DLL spécifiée.

ID d'erreur : BC31014

Pour corriger cette erreur

- Vérifiez que le nom de la procédure est correctement orthographié et que la casse correspond à celle de la fonction dans la DLL.

Voir aussi

Tâches

Procédure pas à pas : appel des API Windows

Impossible d'évaluer l'expression

Unable to evaluate expression

L'expression ne peut pas être évaluée.

ID d'erreur : BC30707

Pour corriger cette erreur

- Vérifiez la syntaxe de l'expression.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Impossible de créer l'assembly : <message d'erreur>

Unable to emit assembly: <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, aussi appelé Alink) pour générer un assembly avec un manifeste, et l'éditeur de liens signale une erreur lors de l'étape d'émission de création de l'assembly.

ID d'erreur : BC30145

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Essayez de signer l'assembly manuellement en utilisant Assembly Linker (Al.exe) ou Outil Strong Name Tool (Sn.exe).
3. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Pour signer l'assembly manuellement

1. Utilisez Outil Strong Name Tool (Sn.exe) pour créer un fichier de paire de clés publique/privée.

   Ce fichier porte une extension .snk.
2. Supprimez la référence COM qui génère l'erreur de votre projet.
3. Dans le menu **Démarrer** de Windows, pointez sur **Programmes**, sur **Microsoft Visual Studio 2005**, sur **Visual Studio Tools**, puis cliquez sur **Invite de commandes Visual Studio 2005**.
4. Déplacez-vous vers le répertoire dans lequel vous souhaitez placer le wrapper d'assembly.
5. Tapez le code ci-dessous.

   ```
   tlbimp <path to COM reference file> /out:<output assembly name> /keyfile:<path to .snk file>
   ```

   Voici un exemple de code que vous pourriez entrer.

   ```
   tlbimp c:\windows\system32\msi.dll /out:Interop.WindowsInstaller.dll /keyfile:"c:\documents and settings\mykey.snk"
   ```

   Utilisez des guillemets doubles (") si un chemin d'accès ou un fichier contient des espaces.
6. Dans Visual Studio, ajoutez une référence d'assembly .NET au fichier que vous venez de créer.

Voir aussi

Tâches

Comment : créer une paire de clés publique/privée

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Outil Strong Name Tool (Sn.exe)

Autres ressources

Support technique et accessibilité

Impossible d'incorporer le fichier de ressources '<nomfichier>' : <message d'erreur>

Unable to embed resource file '<filename>': <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de l'incorporation d'un fichier de ressources COM+ natif directement dans l'assembly.

ID d'erreur : BC30143

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Impossible de créer un fichier de documentation XML '<nom>' : <message>

Unable to create XML documentation file '<name>': <message>

Le fichier de documentation XML ne peut pas être créé en raison d'une erreur.

ID d'erreur : BC42311

Les messages suivants peuvent apparaître avec cette erreur :

- Le bloc de commentaires XML ne peut pas être associé à un élément de langage qui prend en charge l'application des commentaires de la documentation XML.
- La balise de commentaire XML 'returns' n'est pas autorisée pour une propriété 'WriteOnly'.
- Le commentaire XML ne peut pas être appliqué plusieurs fois pour un <type> partiel.
- La balise de commentaire XML 'returns' n'est pas autorisée pour un élément de langage 'declare sub'.
- Erreur d'analyse de la documentation XML : la balise de début '<balise>' n'a pas de balise de fin correspondante.
- Le paramètre de type du commentaire XML <paramètre> ne correspond pas à un paramètre de type pour l'instruction <motclé> correspondante.
- Le paramètre de type de commentaire XML doit avoir un attribut 'name'.
- L'exception de commentaire XML doit avoir un attribut 'cref'.

Pour corriger cette erreur

- Pour des recommandations, consultez le message fourni.

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Impossible de créer le fichier temporaire dans le chemin d'accès '<nomfichier>' : <message d'erreur>

Unable to create temp file in path '<filename>': <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la création d'un fichier ou lors de l'écriture d'une ressource en mémoire dans ce fichier. Il peut s'agir d'un problème de configuration.

ID d'erreur : BC30138

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Impossible de créer un assembly avec nom fort à partir du fichier de clé '<nomfichier>' : <erreur>

Unable to create strong-named assembly from key file '<filename>': <error>

Un assembly avec nom fort n'a pu être créé à partir du fichier de clé spécifié.

ID d'erreur : BC31026

Pour corriger cette erreur

- Vérifiez que le fichier de clé spécifié est correct et qu'il n'est pas verrouillé par une autre application.

Voir aussi

Référence

Outil Strong Name Tool (Sn.exe)

Impossible de créer l'objet Assembly Linker : <message d'erreur>

Unable to create Assembly Linker object: <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la création de son objet de travail. Il peut s'agir d'un problème générant une référence à un fichier de métadonnées ou d'un échec d'appel d'un fichier d'importation.

ID d'erreur : BC30141

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Impossible de créer une interface runtime .NET : <erreur>

Unable to create a .NET Runtime interface: <error>

Une erreur de compilation interne s'est produite lors de la création d'une interface ou un appel de méthode sur une interface a échoué.

ID d'erreur : BC31024

Pour corriger cette erreur

1. Enregistrez votre travail et redémarrez Visual Studio.
2. Si l'erreur se reproduit, réinstallez Visual Studio.
3. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Impossible d'appliquer des attributs de sécurité à '<nom>' : <erreur>

Unable to apply security attribute(s) to '<name>': <error>

Une erreur s'est produite lors de l'application des attributs de sécurité. Celle-ci peut résulter de l'utilisation incorrecte d'un attribut de sécurité ou indiquer un problème avec le compilateur.

ID d'erreur : BC31520

Pour corriger cette erreur

1. Vérifiez la syntaxe et les arguments pour l'attribut.
2. Recompilez le programme afin de vérifier si l'erreur se reproduit.
3. Si cette erreur persiste, enregistrez votre travail et redémarrez Visual Studio.
4. Si l'erreur se reproduit, réinstallez Visual Basic.
5. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Impossible d'accéder au membre

Unable to access member

Le membre spécifié est inaccessible.

ID d'erreur : BC30749

Pour corriger cette erreur

1. Vérifiez que le membre est défini et que vous avez correctement orthographié son nom.
2. Essayez d'accéder à l'un des membres déclarés dans le module. Dans certains cas, l'environnement de débogage ne peut pas localiser les membres car les modules dans lesquels ils sont déclarés n'ont pas encore été chargés.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Les types déclarés 'Private' doivent se trouver à l'intérieur d'un autre type

Types declared 'Private' must be inside another type

Le modificateur **Private** a été utilisé sur un type qui ne se trouve pas à l'intérieur d'un autre type.

ID d'erreur : BC31089

Pour corriger cette erreur

1. Utilisez un modificateur d'accès moins restrictif, tel que **Friend**.
2. Déclarez le type à l'intérieur d'un autre type.

Voir aussi

Référence

Private (Visual Basic)

Les types ne sont pas disponibles dans ce contexte

Types are not available in this context

Vous avez tenté de référencer un type dans une directive **#If**.

ID d'erreur : BC30458

Pour corriger cette erreur

- Supprimez la référence.

Voir aussi

Référence

#If...Then...#Else, directives

Autres ressources

Compilation conditionnelle (Visual Basic)

'TypeOf...Is' requiert un opérande gauche ayant un type référence, mais cet opérande a le type valeur '<type>'

'TypeOf...Is' requires its left operand to have a reference type, but this operand has the type '<type>'

L'expression **TypeOf...Is** vérifie la compatibilité du type d'exécution d'une variable objet. Cette compatibilité n'est pas définie pour les types valeur.

ID d'erreur : BC30021

Pour corriger cette erreur

- Si **Option Strict** a la valeur **Off**, utilisez la fonction **TypeName** ou **VarType** pour obtenir des informations sur les types de données de la variable.
- Si **Option Strict** a la valeur **On**, la déclaration de variable détermine le type de données de la variable.

Voir aussi

Référence

TypeName, fonction (Visual Basic)
VarType, fonction (Visual Basic)
Option Strict, instruction

Concepts

Opérateurs de comparaison en Visual Basic

Les instructions 'Type' ne sont plus prises en charge ; utilisez les instructions 'Structure' à la place

'Type' statements are no longer supported; use 'Structure' statements instead

Le mot clé **Type** ne peut plus être utilisé pour définir les types définis par l'utilisateur.

ID d'erreur : BC30802

Pour corriger cette erreur

- Utilisez l'instruction **Structure** pour définir un nouveau type.

Voir aussi

Référence

Structure, instruction

Autres ressources

Structures constituées à partir de vos propres types de données

Les paramètres de type ou les types construits avec des paramètres de type ne sont pas autorisés dans les arguments d'attribut

Type parameters or types constructed with type parameters are not allowed in attribute arguments

Un attribut est appliqué à l'aide d'un argument qui est un paramètre de type ou est construit à l'aide d'un paramètre de type.

Visual Basic et le .NET Framework ne prennent pas en charge actuellement les combinaisons d'attributs et de types génériques. Cela signifie les restrictions suivantes s'appliquent :

- Un attribut ne peut pas être un type générique ou être déclaré dans un type générique.
- Un attribut ne peut pas hériter d'une classe générique, et inversement.
- Lorsque vous appliquez un attribut, vous ne pouvez pas fournir un argument qui fait partie des éléments suivants :
    - un type générique,
    - un type construit à partir d'un type générique,
    - un paramètre de type d'un type conteneur, ou
    - un type construit à partir d'un paramètre de type d'un type conteneur.

ID d'erreur : BC32079

Pour corriger cette erreur

- Reconstruisez les arguments fournis à l'attribut afin qu'ils ne contiennent pas de paramètres de type ou de type construit à partir d'un paramètre de type.

Voir aussi

Référence

Liste de types
Attribute

Concepts

Vue d'ensemble des attributs dans Visual Basic
Types génériques en Visual Basic

Les paramètres de type ne peuvent pas être utilisés comme qualificateurs

Type parameters cannot be used as qualifiers

Un élément de programmation est qualifié avec une chaîne de qualification qui inclut un paramètre de type.

Un paramètre de type représente une exigence pour un type à fournir lorsque le type générique est construit. Il ne représente pas un type défini spécifique. Une chaîne de qualification ne doit inclure que les éléments définis au moment de la compilation.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Public Function checkText(Of c As System.Windows.Forms.Control) _
    (ByVal badText As String) As Boolean
    Dim saveText As c.Text
    ' Insert code to look for badText within saveText.
End Function
```

ID d'erreur : BC32098

Pour corriger cette erreur

1. Supprimez le paramètre de type de la chaîne de qualification ou remplacez-le par un type défini.
2. Si vous devez utiliser un type construit pour rechercher l'élément de programmation qualifié, vous devez utiliser une logique de programme supplémentaire.

Voir aussi

Tâches

Comment : qualifier un nom d'élément déclaré

Référence

Liste de types

Concepts

Résolution d'une référence lorsque plusieurs variables ont le même nom

Types génériques en Visual Basic

Les paramètres de type ne peuvent pas être spécifiés pour cette déclaration

Type parameters cannot be specified on this declaration

Un élément de programmation est déclaré avec une liste de paramètres de type, mais il ne peut pas être un type générique.

Les éléments de programmation qui ne peuvent pas être génériques sont les propriétés, les opérateurs, les événements et les constructeurs. La déclaration de l'un de ces éléments avec une liste de paramètres de type génère cette erreur.

ID d'erreur : BC32065

Pour corriger cette erreur

- Supprimez le mot clé **Of** et les paramètres de type de la déclaration.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Le paramètre de type avec une contrainte 'Structure' ne peut pas être utilisé en tant que contrainte

Type parameter with a 'Structure' constraint cannot be used as a constraint

Un paramètre de type avec une contrainte **Structure** est utilisé comme contrainte pour un autre paramètre de type.

La contrainte **Structure** requiert que l'argument de type passé à son paramètre de type soit un type valeur. Toutefois, un type valeur ne pouvant pas être implémenté ou hérité, cela n'a pas de sens de l'utiliser comme contrainte, ce qui nécessiterait que l'autre paramètre de type l'implémente ou en hérite.

La seule interprétation utile de cette situation est que les deux arguments de type doivent être exactement du même type. Dans ce cas, votre type générique a besoin d'un seul paramètre de type.

L'instruction suivante peut générer ce message d'erreur.

```
Class c1(Of t1 As Structure, t2 As t1)
```

Le type passé à `t2` ne peut pas implémenter ou hériter du type passé à `t1`, car le type passé à `t1` doit être un type valeur.

ID d'erreur : BC32114

Pour corriger cette erreur

- Supprimez le paramètre de type contraint à **Structure** dans la liste des contraintes de l'autre paramètre de type.
- Si les deux paramètres de type requièrent le même type valeur, définissez le type générique avec un seul paramètre de type.

Voir aussi

Référence

Liste de types

Structure (Visual Basic)

Concepts

Types génériques en Visual Basic

Types valeur et types référence

Le paramètre de type n'est pas autorisé dans la clause 'Implements'

Type parameter not allowed in 'Implements' clause

Une clause **Implements** dans un type générique spécifie un paramètre de type comme membre à implémenter.

Une clause **Implements** doit spécifier une interface et un membre. Elle peut passer un paramètre de type à l'interface, mais elle ne peut pas le passer au membre, ni l'utiliser comme nom du membre.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Class c1(Of t)
    Implements i1(Of t)
    Public Sub doSomething() Implements t
End Class
```

ID d'erreur : BC32056

Pour corriger cette erreur

- Spécifiez le nom d'interface et un membre authentique de l'interface qui suit le mot clé **Implements**. Vous pouvez passer le paramètre de type à l'interface, si besoin.

  ```
      Public Sub doSomething() Implements i1(Of t).doSomething
  ```

Voir aussi

Référence

Implements (Visual Basic)
Liste de types

Concepts

Implements, mot clé et instruction
Types génériques en Visual Basic

Le nom du paramètre de type '<NomParamètreDeType1>' ne correspond pas au nom '<NomParamètreDeType2>' du paramètre de type correspondant défini pour l'un des autres types partiels de '<NomTypePartiel>'

Type parameter name '<typeparametername1>' does not match the name '<typeparametername2>' of the corresponding type parameter defined on one of the other partial types of '<partialtypename>'

Une classe ou une structure générique est définie dans plusieurs déclarations partielles avec des caractéristiques de paramètre de type incompatibles.

Lorsque vous divisez la définition d'une classe ou d'une structure en plusieurs déclarations partielles, le compilateur traite le type comme l'union de toutes ses déclarations partielles. Cela s'applique non seulement aux membres, mais également à l'implémentation, l'héritage et le niveau d'accès.

Vous ne pouvez pas spécifier plusieurs noms pour les paramètres de type dans la définition d'une classe ou d'une structure générique.

ID d'erreur : BC30931

Pour corriger cette erreur

- Définissez le nom du paramètre de type et utilisez le même nom dans chaque déclaration partielle.

Voir aussi

Référence

Partial (Visual Basic)
Class, instruction (Visual Basic)
Structure, instruction
Liste de types

Concepts

Classes : modèles d'objets
Types génériques en Visual Basic

Autres ressources

Structures constituées à partir de vos propres types de données

Un paramètre de type ne peut pas avoir le même nom que sa fonction de définition

Type parameter cannot have the same name as its defining function

Un paramètre de type d'un type générique est déclaré avec le même nom que le type générique.

Dans la liste des paramètres de type d'un type générique, chaque paramètre de type doit avoir un nom différent de tous les noms suivants :

- Tous les autres paramètres de type contenus dans la même liste des paramètres de type,
- Tous les paramètres de type contenus dans la liste des paramètres de type d'un type générique conteneur, et
- Nom du type générique proprement dit.

ID d'erreur : BC32090

Pour corriger cette erreur

- Renommez le paramètre de type pour que son nom soit différent des noms cités dans la liste de cette page d'aide.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Le paramètre de type est déjà déclaré avec le nom '<NomParamètreDeType>'

Type parameter already declared with name '<typeparametername>'

Un paramètre de type d'un type générique est déclaré avec le même nom qu'un autre paramètre de type de ce même type générique.

Dans la liste des paramètres de type d'un type générique, chaque paramètre de type doit avoir un nom différent des noms suivants :

- tout autre paramètre de type présent dans la même liste de paramètres de type ;
- chaque paramètre de type présent dans la liste des paramètres d'un type générique conteneur et ;
- nom du type générique lui-même.

ID d'erreur : BC32049

Pour corriger cette erreur

- Renommez le paramètre de type à distinguer de chaque nom cité dans la liste de cette page d'aide.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Le paramètre de type '<NomParamètreDeType1>' doit avoir une contrainte 'New' ou une contrainte 'Structure' pour satisfaire la contrainte 'New' pour le paramètre de type '<NomParamètreDeType2>'

Type parameter '<typeparametername1>' must have either a 'New' constraint or a 'Structure' constraint to satisfy the 'New' constraint for type parameter '<typeparametername2>'

Une instruction construit un type générique en passant un paramètre de type qui n'est pas contraint pour satisfaire une contrainte **New**.

La contrainte **New** signifie que l'argument de type fourni à ce paramètre de type doit exposer un constructeur sans paramètre accessible au code qui crée des objets à partir de celui-ci. Tous les types valeur ont des constructeurs sans paramètres, mais pas tous les types référence. Par conséquent, la contrainte **Structure** satisfait la contrainte **New**, mais pas la contrainte **Class**, ou un nom de classe ou d'interface.

Les instructions suivantes peuvent générer ce message d'erreur.

```
Public Class c1(Of t As New)
End Class
Public Class c2(Of u)
    Public testObject As New c1(Of u)
End Class
```

Lorsque la classe `c2` crée un objet de `c1`, le paramètre de type `u` ne satisfait pas la contrainte **New** sur le paramètre de type `t`.

ID d'erreur : BC32084

Pour corriger cette erreur

- Si le paramètre de type à passer au type générique peut satisfaire la contrainte **Structure** ou **New**, ajoutez la contrainte appropriée à sa définition.

  ```
  Public Class c2(Of u As Structure)
  ```

- Si le paramètre de type ne peut pas satisfaire la contrainte **Structure** ou **New**, ne le passez pas au type générique. Vous devez passer un autre élément comme argument de type.

Voir aussi

Référence

New (Visual Basic)
Structure (Visual Basic)
Class (Visual Basic)

Concepts

Types génériques en Visual Basic
Types valeur et types référence

Le paramètre de type '<NomParamètreDeType>' ne peut pas être contraint par lui-même : '<MessageErreur>'

Type parameter '<typeparametername>' cannot be constrained to itself: '<errormessage>'

Une liste de contraintes pour un paramètre de type contient ce même paramètre de type.

Une liste de contraintes sur un paramètre de type peut spécifier un nombre quelconque d'interfaces et au plus une classe. Un argument de type fourni pour ce paramètre de type doit implémenter chaque interface spécifiée et hériter de la classe spécifiée. Le compilateur requiert des interfaces et des classes qui sont déjà définies lorsqu'il rencontre une liste de contraintes. Un paramètre de type n'est pas considéré comme un type défini tant qu'il n'est pas remplacé par un argument de type convenable fourni par le code qui crée le type générique.

ID d'erreur : BC32113

Pour corriger cette erreur

1. Vérifiez l'orthographe du paramètre de type et des contraintes dans sa liste de contraintes.
2. S'il existe des fautes d'orthographe, supprimez le nom du paramètre de type de sa liste de contraintes. Il ne peut pas être contraint par lui-même.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Le paramètre de type '<NomParamètreDeType>' ne peut avoir qu'une seule contrainte qui est une classe

Type parameter '<typeparametername>' can only have one constraint that is a class

Une liste de contraintes contient plusieurs classes.

Une liste de contraintes sur un paramètre de type peut accepter un nombre quelconque d'interfaces, mais au plus une classe. Un argument de type fourni pour ce paramètre de type doit hériter de cette classe, et Visual Basic ne prend pas en charge l'héritage multiple.

ID d'erreur : BC32047

Pour corriger cette erreur

- Sélectionnez une classe et incluez uniquement cette classe dans la liste de contraintes.
- Vous pouvez définir des paramètres de type supplémentaires pour prendre en charge la ou les classes qui n'ont pas été incluses dans cette liste de contraintes.

Voir aussi

Concepts

Types génériques en Visual Basic

Le paramètre de type '<NomParamètreDeType>' a le même nom qu'un paramètre de type d'un type englobant

Type parameter '<typeparametername>' has the same name as a type parameter of an enclosing type

Un paramètre de type d'un type générique est déclaré avec le même nom qu'un paramètre de type d'un type générique conteneur.

Dans la liste des paramètres de type d'un type générique, chaque paramètre de type doit avoir un nom différent des noms suivants :

- tout autre paramètre de type présent dans la même liste de paramètres de type ;
- chaque paramètre de type présent dans la liste des paramètres d'un type générique conteneur et ;
- nom du type générique lui-même.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40048

Pour corriger cette erreur

- Renommez le paramètre de type à distinguer de chaque nom cité dans la liste de cette page d'aide.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Le paramètre de type '<NomParamètreDeType>' pour '<NomProcédureGénérique>' ne peut pas être inféré

Type parameter '<typeparametername>' for '<genericprocedurename>' cannot be inferred

Une procédure générique est appelée sans fournir une liste d'arguments de type, et l'inférence du type échoue pour l'un des arguments de type.

Lorsque vous appelez une procédure générique, vous fournissez normalement un argument de type pour chaque paramètre de type défini par la procédure. Toutefois, vous avez la possibilité d'omettre totalement la liste d'arguments de type. Si vous le faites, le compilateur essaie de déduire le type de chaque argument de type à partir du contexte de votre appel. Pour plus d'informations, consultez « Inférence du type » dans Procédures génériques dans Visual Basic.

Une incompatibilité de rang entre un paramètre de type et le type appelant peut expliquer l'échec de l'inférence du type. Le code suivant illustre ce comportement :

```
Public Sub displayLargest(Of t As IComparable)(ByVal arg() As t)
    ' Insert code to find and display the largest element of arg().
End Sub
Public Sub callGenericSub()
    Dim testValue As Integer
    findLargest(testValue)
    Dim testMatrix(,) As Integer
    findLargest(testMatrix)
End Sub
```

Dans le code précédent, les deux appels à `findLargest` produisent cette erreur, parce que le paramètre de type `t` fait appel à un tableau unidimensionnel, alors que les arguments de type que le compilateur déduit à partir des appels représentent un tableau scalaire (`testValue`) et en deux dimensions (`testMatrix`).

ID d'erreur : BC32050

Pour corriger cette erreur

- Vérifiez que les types des arguments normaux se présentent de telle sorte que l'inférence du type soit cohérente par rapport aux paramètres de type déclarés pour la procédure générique.

  - ou -

- Appelez la procédure générique avec une liste d'arguments de type complète pour que l'inférence du type ne soit pas nécessaire.

Voir aussi

Référence

Liste de types

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Type ou 'New' attendu

Type or 'New' expected

Un paramètre de type dans la déclaration d'un type générique introduit une liste de contraintes avec le mot clé **As**, mais ne spécifie pas de contrainte valide.

Une contrainte sur un paramètre de type doit être une classe ou une interface valide, ou l'un des mots clés **Class**, **Structure** ou **New**. Si vous spécifiez une contrainte non valide ou aucun contrainte, le compilateur génère cette erreur.

ID d'erreur : BC32092

Pour corriger cette erreur

1. Déterminez la manière dont le paramètre de type doit être contraint et spécifiez la contrainte appropriée dans la liste des contraintes.
2. Si vous comptez appliquer une contrainte au paramètre de type par une classe ou une interface, vérifiez que la contrainte est correctement orthographiée.
3. N'oubliez pas de séparer par des virgules plusieurs contraintes sur un paramètre de type unique, et de mettre la liste des contraintes entre accolades (**{ }**).

Voir aussi

Référence

Class (Visual Basic)
Structure (Visual Basic)
New (Visual Basic)
Liste de types

Concepts

Types génériques en Visual Basic

Le type de paramètre '<NomParamètre>' n'est pas conforme CLS

Type of parameter '<parametername>' is not CLS-compliant

Une procédure est marquée comme `<CLSCompliant(True)>` mais déclare un paramètre avec un type marqué comme `<CLSCompliant(False)>`, non marqué ou non qualifié à cause de sa non conformité.

Pour qu'une procédure soit conforme Spécification CLS (Common Language Specification) (CLS), elle ne doit utiliser que des types conformes CLS. Cette règle s'applique aux types des paramètres, au type de retour et aux types de toutes ses variables locales.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

Lorsque vous appliquez CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40028

Pour corriger cette erreur

- Si la procédure doit prendre un paramètre de ce type particulier, supprimez l'**CLSCompliantAttribute**. La procédure ne peut pas être conforme CLS.
- Si la procédure doit être conforme CLS, remplacez le type de ce paramètre par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.
- Si vous utilisez des objets Automation ou COM, gardez à l'esprit que certains types ont des largeurs des données différentes de celles du .NET Framework. Par exemple, **int** correspond souvent à 16 bits dans d'autres environnements. Si vous passez un argument de 16 bits à un tel composant, déclarez-le comme type de données **Short** et non comme **Integer** dans votre code managé Visual Basic.

Voir aussi

Concepts

Écriture d'un code conforme CLS

Le type de la valeur facultative pour le paramètre optionnel <NomParamètre> n'est pas conforme CLS

Type of optional value for optional parameter <parametername> is not CLS-compliant

Une procédure est marquée comme `<CLSCompliant(True)>`, mais déclare un paramètre Optional (Visual Basic) avec une valeur par défaut d'un type non conforme.

Pour qu'une procédure soit conforme au Spécification CLS (Common Language Specification) (CLS), elle doit utiliser uniquement des types conformes CLS. Cela s'applique aux types des paramètres, au type de retour et aux types de toutes ses variables locales. Cela s'applique également aux valeurs par défaut des paramètres optionnels.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

Lorsque vous appliquez l'attribut CLSCompliantAttribute à un élément de programmation, vous affectez au paramètre *isCompliant* de l'attribut la valeur **True** ou **False** pour indiquer la conformité ou la non-conformité. Il n'existe pas de valeur par défaut pour ce paramètre et vous devez fournir une valeur.

Si vous n'appliquez pas **CLSCompliantAttribute** à un élément, il est considéré comme étant non conforme.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40042

Pour corriger cette erreur

- Si le paramètre optionnel doit avoir une valeur par défaut de ce type particulier, supprimez **CLSCompliantAttribute**. La procédure ne peut pas être conforme CLS.
- Si la procédure doit être conforme CLS, remplacez le type de cette valeur par défaut par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.
- Si vous utilisez des objets Automation ou COM, n'oubliez pas que certains types ont des largeurs de données différentes de celles le .NET Framework. Par exemple, **int** correspond souvent à 16 bits dans d'autres environnements. Si vous acceptez un argument de 16 bits à un tel composant, déclarez-le comme type de données **Short** et non comme **Integer** dans votre code Visual Basic managé.

Voir aussi

Concepts

Écriture d'un code conforme CLS

Le type de membre '<NomMembre>' n'est pas conforme CLS

Type of member '<membername>' is not CLS-compliant

Le type de données spécifié pour ce membre ne fait pas partie du Spécification CLS (Common Language Specification) (CLS). Cette erreur ne provient pas de votre composant, parce que le .NET Framework et Visual Basic prennent en charge ce type de données. Toutefois, il est possible qu'un autre composant écrit dans le code strictement conforme CLS ne prenne pas en charge ce type de données. Ce composant peut ne pas interagir avec succès avec votre composant.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40025

Pour corriger cette erreur

- Si votre composant sert d'interface uniquement avec d'autres composants du .NET Framework, ou ne sert pas d'interface avec d'autres composants, aucune modification n'est nécessaire.
- Si vous utilisez un composant non écrit pour le .NET Framework, vous pouvez déterminer, par réflexion ou à partir de la documentation, s'il prend en charge ce type de données. Si tel est le cas, aucune modification n'est nécessaire.
- Si vous utilisez un composant qui ne prend pas en charge ce type de données, vous devez le remplacer par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.
- Si vous utilisez des objets Automation ou COM, n'oubliez pas que certains types ont des largeurs de données différentes de celles dans le .NET Framework. Par exemple, **uint** correspond souvent à 16 bits dans d'autres environnements. Si vous passez un argument de 16 bits à un tel composant, déclarez-le comme type de données **UShort** et non comme **UInteger** dans votre code Visual Basic managé.

Voir aussi

Concepts

Vue d'ensemble de la réflexion
Écriture d'un code conforme CLS

Autres ressources

Réflexion

Impossible de charger la bibliothèque référencée '<nomfichier>' : <erreur>

Unable to load referenced library '<filename>': <error>

Impossible de charger une bibliothèque référencée par ce projet.

ID d'erreur : BC31011

Pour corriger cette erreur

- Vérifiez que le fichier se trouve à l'emplacement spécifié et qu'il n'est pas verrouillé par une autre application.

Voir aussi

Concepts

Références de projet

Impossible de charger les informations pour la classe '<nomclasse>'

Unable to load information for class '<classname>'

Une référence a été faite à une classe non disponible.

ID d'erreur : BC30712

Pour corriger cette erreur

1. Vérifiez que la classe est définie et que vous avez correctement orthographié son nom.
2. Essayez d'accéder à l'un des membres déclarés dans le module. Dans certains cas, l'environnement de débogage ne peut pas localiser les membres car les modules dans lesquels ils sont déclarés n'ont pas encore été chargés.

Voir aussi

Concepts

Classes : modèles d'objets

Autres ressources

Débogage dans Visual Studio

Impossible de charger la DLL '<nomfichier>' : <erreur>

Unable to load DLL '<filename>': <error>

Impossible de charger une DLL référencée par ce projet.

ID d'erreur : BC31013

Pour corriger cette erreur

- Vérifiez que le fichier se trouve à l'emplacement spécifié et qu'il n'est pas verrouillé par une autre application.

Voir aussi

Concepts

Types et extensions de fichiers en Visual Basic, Visual C# et Visual J#
Références de projet

Impossible de créer un lien vers le fichier de ressources '<nomfichier>' : <message d'erreur>

Unable to link to resource file '<filename>': <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la liaison à un fichier de ressources COM+ natif à partir de l'assembly.

ID d'erreur : BC30144

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Impossible d'obtenir des informations de type pour '<nomtype>'

Unable to get type information for '<typename>'

Un type n'ayant pas été chargé par le Common Language Runtime a été référencé.

ID d'erreur : BC30751

Pour corriger cette erreur

- Restructurez le code de manière à ce que le type soit défini dans la portée actuelle.

Voir aussi

Autres ressources

Débogage dans Visual Studio

Impossible d'analyser le code XML : <erreur>

Unable to parse XML: <error>

Une erreur existe dans le fichier XML utilisé.

ID d'erreur : BC31023

Pour corriger cette erreur

- Vérifiez que le code XML présente une forme correcte, est valide et que le fichier n'est pas endommagé.

Voir aussi

Autres ressources

Documents et données XML

Impossible d'ouvrir le fichier réponse '<nomfichier>'

Unable to open response file '<filename>'

Le fichier auquel vous tentez d'accéder n'est pas disponible.

ID d'erreur : BC2011

Pour corriger cette erreur

- Assurez-vous que votre connexion réseau est toujours active.
- Vérifiez vos autorisations sur le fichier.

Voir aussi

Référence

FileIOPermission

Concepts

Types d'erreurs

Impossible d'ouvrir le fichier de ressources '<nomfichier>' : <erreur>

Unable to open resource file '<filename>': <error>

Une erreur s'est produite lors de l'ouverture du fichier de ressources spécifié.

ID d'erreur : BC31509

Pour corriger cette erreur

1. Vérifiez que le nom et l'emplacement spécifiés sont corrects.
2. Vérifiez que le fichier n'est pas endommagé ou corrompu.

Voir aussi

Référence

/linkresource (Visual Basic)
/resource (Visual Basic)

Impossible d'ouvrir le fichier de module '<nomfichier>' : <erreur>

Unable to open module file '<filename>': <error>

Impossible d'ouvrir un fichier contenant un module de ce projet.

ID d'erreur : BC31007

Pour corriger cette erreur

- Vérifiez que le fichier se trouve à l'emplacement spécifié et qu'il n'est pas verrouillé par une autre application.

Voir aussi

Concepts

Types et extensions de fichiers en Visual Basic, Visual C# et Visual J#

Impossible d'ouvrir le fichier de clé '<nomfichier>' : <erreur>

Unable to open key file '<filename>': <error>

Impossible d'ouvrir le fichier de clé spécifié pour ce projet.

ID d'erreur : BC31025

Pour corriger cette erreur

- Vérifiez que le fichier de clé existe et qu'il n'est pas verrouillé par une autre application.

Voir aussi

Référence

Outil Strong Name Tool (Sn.exe)

Concepts

Attributs globaux en Visual Basic

Impossible d'ouvrir le fichier '<nomfichier>' : <erreur>

Unable to open file '<filename>': <error>

Une erreur s'est produite lors de l'ouverture d'un fichier de ce projet.

ID d'erreur : BC31027

Pour corriger cette erreur

- Vérifiez que le fichier existe, qu'il n'est pas endommagé ou verrouillé par une autre application.

Voir aussi

Concepts

Types et extensions de fichiers en Visual Basic, Visual C# et Visual J#

Impossible de signer le fichier '<nomfichier>' : <erreur>

Unable to sign file '<filename>': <error>

Une erreur est survenue lors de la tentative de signature du fichier spécifié. Cette erreur a pu se produire pour plusieurs raisons :

- Autorisations insuffisantes.
- Fichiers système requis pour la signature Authenticode manquants.
- Référence à un certificat ou à un fichier de clé privée inexistant.
- Orthographe incorrecte d'un nom de fichier ou d'une URL.

ID d'erreur : BC31028

Pour corriger cette erreur

1. Entrez des noms corrects de certificats et de fichiers de clés privées.
2. Si vous utilisez la signature Authenticode, vérifiez que les fichiers Signcode.exe et Javasign.dll existent et qu'ils ne sont pas en lecture seule.
3. Assurez-vous que vous disposez de l'autorisation **Write** sur le fichier.

Voir aussi

Référence

Outil File Signing Tool (Signcode.exe)

Concepts

Déploiement et signature Authenticode

Impossible de signer l'assembly : <message d'erreur>

Unable to sign assembly: <error message>

Le compilateur Visual Basic appelle l'utilitaire Assembly Linker (Al.exe, également connu sous le nom d'Alink) pour générer un assembly avec un manifeste. L'éditeur de liens a signalé une erreur lors de la signature de l'assembly, après sa création. Cela peut être dû à une clé non valide.

ID d'erreur : BC30146

Pour corriger cette erreur

1. Examinez le message d'erreur entre guillemets et consultez la rubrique Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings) pour obtenir davantage d'explications et de conseils.
2. Si l'erreur persiste, regroupez les informations sur les circonstances et avertissez les services de Support Technique de Microsoft.

Voir aussi

Référence

Assembly Linker (Al.exe)

Erreurs et avertissements de l'outil Al.exe (Al.exe Tool Errors and Warnings)

Autres ressources

Support technique et accessibilité

Impossible d'écrire dans un fichier temporaire, car le chemin d'accès temporaire n'est pas disponible

Unable to write temporary file because temporary path is not available

Visual Basic n'a pas pu déterminer le chemin d'accès de l'emplacement de stockage des fichiers temporaires.

ID d'erreur : BC30698

Pour corriger cette erreur

1. Redémarrez Visual Studio.
2. Si le problème persiste, réinstallez Visual Studio.

Voir aussi

Autres ressources

Support technique

Impossible d'écrire la sortie dans la mémoire

Unable to write output to memory

Un problème est survenu lors de l'écriture de la sortie dans la mémoire.

ID d'erreur : BC31020

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si cette erreur persiste, enregistrez votre travail et redémarrez Visual Studio.
3. Si l'erreur se reproduit, réinstallez Visual Basic.
4. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Variable locale inutilisée : '<NomVariableLocale>'

Unused local variable: '<localvariablename>'

Une variable locale dans une procédure est déclarée, mais n'est jamais utilisée.

Une explication possible est qu'une faute d'orthographe figure dans les variables locales de la procédure. Si cette variable est utilisée en fait dans une autre instruction mais orthographiée différemment, elle apparaît au compilateur sous la forme de deux variables différentes.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42024

Pour corriger cette erreur

1. Vérifiez les fautes d'orthographe dans les variables locales de la procédure. Notez que la casse importe peu. Les noms `ABC` et `abc` sont considérés comme faisant référence à la même variable.

2. S'il n'existe aucune faute d'orthographe, supprimez la déclaration de cette variable ou utilisez-la dans une autre instruction de la procédure.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Noms d'éléments déclarés

Fibre non planifiée

Unscheduled Fiber

Le débogueur ne peut pas évaluer une expression car elle est présente dans une fibre logique qui n'est pas planifiée sur un thread physique. Cela peut se produire si le processus s'exécute sur un serveur SQL à l'aide de fibres.

Une fibre se compose d'une pile et d'un contexte de registre, et elle peut s'exécuter sur n'importe quel thread. Elle peut être transférée hors d'un thread et redémarrée ultérieurement sur un autre thread.

ID d'erreur : BC30948

Pour corriger cette erreur

- Vérifiez que cette fibre est planifiée sur un thread physique.

Voir aussi

Concepts

Débogage de SQL

Autres ressources

Débogage dans Visual Studio

Option <nomoption> non reconnue ignorée

Unrecognized option <optionname>; ignored

Une option non valide a été spécifiée à partir de la ligne de commande.

ID d'erreur : BC2007

Pour corriger cette erreur

- Vérifiez les options spécifiées pour rechercher celle qui n'est pas valide.

Voir aussi

Référence

Switch, fonction

Le type sous-jacent <NomType> d'Enum n'est pas conforme CLS

Underlying type <typename> of Enum is not CLS-compliant

Le type de données spécifié pour cette énumération ne fait pas partie du Spécification CLS (Common Language Specification) (CLS). Cette erreur ne provient pas de votre composant, parce que le .NET Framework et Visual Basic prennent en charge ce type de données. Toutefois, il est possible qu'un autre composant écrit dans le code strictement conforme CLS ne prenne pas en charge ce type de données. Ce composant peut ne pas interagir avec succès avec votre composant.

Les types de données Visual Basic suivants ne sont pas conformes CLS :

- SByte, type de données (Visual Basic)
- UInteger, type de données
- ULong, type de données (Visual Basic)
- UShort, type de données (Visual Basic)

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC40032

Pour corriger cette erreur

- Si votre composant sert d'interface uniquement avec d'autres composants du .NET Framework, ou ne sert pas d'interface avec d'autres composants, aucune modification n'est nécessaire.
- Si vous utilisez un composant non écrit pour le .NET Framework, vous pouvez déterminer, par réflexion ou à partir de la documentation, s'il prend en charge ce type de données. Si tel est le cas, aucune modification n'est nécessaire.
- Si vous utilisez un composant qui ne prend pas en charge ce type de données, vous devez le remplacer par le type conforme CLS le plus proche. Par exemple, vous pouvez utiliser **Integer** au lieu de **UInteger** si vous n'avez pas besoin de la plage de valeurs située au-dessus de 2 147 483 647. Si vous avez besoin de la plage étendue, vous pouvez remplacer **UInteger** par **Long**.
- Si vous utilisez des objets Automation ou COM, n'oubliez pas que certains types ont des largeurs de données différentes de celles dans le .NET Framework. Par exemple, **uint** correspond souvent à 16 bits dans d'autres environnements. Si vous passez un argument de 16 bits à un tel composant, déclarez-le comme type de données **UShort** et non comme **UInteger** dans votre code Visual Basic managé.

Voir aussi

Concepts

Vue d'ensemble de la réflexion
Écriture d'un code conforme CLS

Autres ressources

Réflexion

Impossible d'écrire dans le fichier de sortie '<nomfichier>' : <erreur>

Unable to write to output file '<filename>': <error>

Un problème est survenu lors de la création du fichier.

Un fichier de sortie ne peut pas être ouvert pour écriture. Le fichier (ou le dossier qui le contient) peut être ouvert en mode exclusif par un autre processus ou bien il peut avoir son propre jeu d'attributs en lecture seule.

Il existe plusieurs situations courantes dans lesquelles un fichier est ouvert exclusivement :

- L'application est déjà en cours d'exécution et utilise ses fichiers. Pour résoudre ce problème, assurez-vous que l'application n'est pas en cours d'exécution.
- Une autre application a ouvert le fichier. Pour résoudre ce problème, assurez-vous qu'aucune autre application n'accède aux fichiers. Il n'est pas toujours évident de déterminer l'application qui accède à vos fichiers ; dans ce cas, le moyen le plus simple pour terminer l'application peut être de redémarrer l'ordinateur.

Même si l'un des fichiers de sortie du projet est marqué en lecture seule, cette exception sera levée.

ID d'erreur : BC31019

Pour corriger cette erreur

1. Recompilez le programme afin de vérifier si l'erreur se reproduit.
2. Si cette erreur persiste, enregistrez votre travail et redémarrez Visual Studio.
3. Si l'erreur persiste, redémarrez l'ordinateur.
4. Si l'erreur se reproduit, réinstallez Visual Basic.
5. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Pour vérifier les attributs de fichier dans l'Explorateur Windows

1. Ouvrez le dossier qui vous intéresse.
2. Cliquez sur l'icône **Affichage** et sélectionnez **Détails**.
3. Cliquez avec le bouton droit sur l'en-tête de colonne, puis sélectionnez **Attributs** dans la liste déroulante.

Pour modifier les attributs d'un fichier ou d'un dossier

1. Dans l'**Explorateur Windows**, cliquez avec le bouton droit sur le fichier ou le dossier, puis sélectionnez **Propriétés**.
2. Dans la section **Attributs** de l'onglet **Général**, désactivez la case à cocher **Lecture seule**.
3. Cliquez sur **OK**.

Voir aussi

Autres ressources

Support technique et accessibilité

'Using' doit se terminer par un 'End Using' correspondant

'Using' must end with a matching 'End Using'

Une instruction **Using** apparaît sans instruction **End Using** correspondante.

Une instruction **End Using** doit être utilisée pour terminer le bloc **Using**.

ID d'erreur : BC36008

Pour corriger cette erreur

1. Si ce bloc **Using** fait partie d'un ensemble de blocs **Using** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Ajoutez une instruction **End Using** à la fin du bloc **Using**.

Voir aussi

Tâches

Comment : supprimer une ressource système

Référence

Using, instruction (Visual Basic)

L'opérande 'Using' de type '<NomType>' doit implémenter 'System.IDisposable'

'Using' operand of type '<typename>' must implement System.IDisposable

Une instruction **Using** spécifie une ressource d'un type qui n'implémente pas l'interface IDisposable.

Un bloc **Using** a pour but de garantir la suppression d'une ressource système à la sortie du bloc. Pour y parvenir, la ressource doit exposer la méthode Dispose implémentée à partir de **IDisposable**.

ID d'erreur : BC36010

Pour corriger cette erreur

- Supprimez la ressource dans la liste des ressources de l'instruction **Using** ou remplacez-la par une ressource qui implémente **IDisposable**.

Voir aussi

Tâches

Comment : supprimer une ressource système

Référence

Using, instruction (Visual Basic)

IDisposable

La variable de ressource 'Using' doit avoir une initialisation explicite

'Using' resource variable must have an explicit initialization

Une instruction **Using** spécifie au moins une ressource qu'elle n'initialise pas avec une clause **New**.

Si vous n'avez pas déjà acquis la ressource avant de passer le contrôle au bloc **Using**, l'instruction **Using** doit acquérir la ressource. Pour ce faire, elle doit créer un objet à partir de la classe spécifiée, qui requiert une clause **New**.

ID d'erreur : BC36011

Pour corriger cette erreur

- Si vous avez déjà acquis la ressource, utilisez une variable ou une expression de référence dans l'instruction **Using** qui a pour valeur la ressource acquise.

```
Dim newFont As New System.Drawing.Font

Using newFont
```

- Si vous n'avez pas déjà acquis la ressource, ajoutez une clause **New** à l'instruction **Using**.

```
Using newFont as New System.Drawing.Font
```

Voir aussi

Tâches

Comment : supprimer une ressource système

Référence

Using, instruction (Visual Basic)

New (Visual Basic)

Le type de variable de ressource 'Using' ne peut pas être de type tableau

'Using' resource variable type can not be array type

Une instruction **Using** spécifie une variable de tableau pour une ressource.

La classe Array n'implémente pas l'interface IDisposable, donc le bloc **Using** ne peut pas appeler la méthode Dispose sur une ressource de tableau.

ID d'erreur : BC36012

Pour corriger cette erreur

- Utilisez un type de ressource système différent dans le bloc **Using**.

Voir aussi

Tâches

Comment : supprimer une ressource système

Référence

Using, instruction (Visual Basic)

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Les instructions 'Using' ne sont pas valides dans la fenêtre Exécution

'Using' statements are not valid in the Immediate window

Les instructions **Using** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30186

Pour corriger cette erreur

- Supprimez l'instruction **Using** de la fenêtre **Exécution**.

Voir aussi

Référence

Using, instruction (Visual Basic)

Fenêtre Exécution

Autres ressources

Débogage dans Visual Studio

La valeur '<nomvaleur1>' ne peut pas être convertie en '<nomvaleur2>'

Value '<valuename1>' cannot be converted to '<valuename2>'

Une tentative de conversion de type a été effectuée entre deux types incompatibles.

ID d'erreur : BC30742

Pour corriger cette erreur

- Vérifiez les types sur lesquels un cast a été effectué.

Voir aussi

Référence

CType, fonction

La valeur de type '<NomType1>' ne peut pas être convertie en '<NomType2>'

Message d'erreur
La valeur de type '<NomType1>' ne peut pas être convertie en '<NomType2>'. L'incompatibilité de type peut résulter de la combinaison d'une référence de fichier et d'une référence de projet pour l'assembly '<NomAssembly>'. Essayez de remplacer la référence de fichier pour '<CheminFichier>' dans le projet '<NomProjet1>' par une référence de projet pour '<NomProjet2>'.
Value of type '<typename1>' cannot be converted to '<typename2>'. Type mismatch could be due to the mixing of a file reference with a project reference to assembly '<assemblyname>'. Try replacing the file reference to '<filepath>' in project '<projectname1>' with a project reference to '<projectname2>'.

Lorsqu'un projet crée une référence de projet et une référence de fichier, le compilateur ne peut pas garantir qu'un type soit converti en un autre.

Le pseudo-code suivant illustre une situation qui peut générer cette erreur.

```
' ================ Visual Basic project P1 ================

' P1 makes a PROJECT REFERENCE to project P2

' and a FILE REFERENCE to project P3.

Public commonObject As P3.commonClass

commonObject = P2.getCommonClass()

' ================ Visual Basic project P2 ================

' P2 makes a PROJECT REFERENCE to project P3

Public Function getCommonClass() As P3.commonClass

Return New P3.commonClass

End Function

' ================ Visual Basic project P3 ================

Public Class commonClass

End Class
```

Le projet `P1` crée une référence de projet indirecte par l'intermédiaire du projet `P2` pour le projet `P3`, et une référence de fichier directe pour `P3`. La déclaration de `commonObject` utilise la référence de fichier pour `P3`, alors que l'appel à `P2.getCommonClass` utilise la référence de projet pour `P3`.

Le problème dans ce cas est que la référence de fichier spécifie un chemin d'accès et un nom pour le fichier de sortie de `P3` (en général, p3.dll), alors que les références de projet identifient le projet source (`P3`) par le nom du projet. De ce fait, le compilateur ne peut pas garantir que le type `P3.commonClass` provienne du même code source pour les deux références différentes.

Cette situation se produit généralement lorsque les références de projet et les références de fichier sont associées. Dans l'illustration précédente, le problème ne se produirait pas si `P1` crée une référence de projet directe pour `P3` au lieu d'une référence de fichier.

ID d'erreur : BC30955

Pour corriger cette erreur

- Remplacez la référence de fichier par une référence de projet.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Références de projet
Autres ressources
Conversions de type en Visual Basic
Références aux espaces de noms et aux composants
Gestion des références

La valeur du type '<NomType1>' ne peut pas être convertie en '<NomType2>' (plusieurs références de fichier)

Message d'erreur
La valeur du type '<NomType1>' ne peut pas être convertie en '<NomType2>'. L'incompatibilité de type peut résulter de la combinaison d'une référence de fichier pour '<CheminFichier1>' dans le projet '<NomProjet1>' et d'une référence de fichier pour '<CheminFichier2>' dans le projet '<NomProjet2>' Si les deux assemblys sont identiques, essayez de remplacer ces deux références pour qu'elles se situent dans le même emplacement.
Value of type '<typename1>' cannot be converted to '<typename2>'. Type mismatch could be due to mixing a file reference to '<filepath1>' in project '<projectname1>' with a file reference to '<filepath2>' in project '<projectname2>'. If both assemblies are identical, try replacing these references so both references are from the same location.

Si un projet fait plusieurs références de fichier à un assembly, le compilateur ne peut pas garantir qu'un type peut être converti en un autre.

Chaque référence de fichier spécifie un chemin d'accès et un nom pour le fichier de sortie d'un projet (en général un fichier DLL). Le compilateur ne peut pas garantir que les fichiers de sortie viennent de la même source ou qu'ils représentent la même version du même assembly. Par conséquent, il ne peut pas garantir que les types dans les références différentes sont du même type ou même que l'un peut être converti en l'autre.

Vous pouvez utiliser une référence de fichier unique si vous savez que les assemblys référencés ont la même identité d'assembly. L'*identité d'assembly* inclut le nom de l'assembly, la version, la clé publique le cas échéant, et la culture. Ces informations identifient de manière unique l'assembly.

ID d'erreur : BC30961

Pour corriger cette erreur

- Si les assemblys référencés ont la même identité d'assembly, supprimez ou remplacez l'une des références de fichier pour qu'il y ait une référence de fichier unique.
- Si les assemblys référencés n'ont pas la même identité d'assembly, modifiez votre code afin qu'il ne tente pas de convertir un type dans l'une en un type dans l'autre.

Voir aussi

Tâches
Comment : ajouter ou supprimer des références en Visual Studio
Concepts
Références de projet
Autres ressources
Conversions de type en Visual Basic
Références aux espaces de noms et aux composants
Gestion des références

Une valeur de type '<type1>' ne peut pas être convertie en '<type2>'

Value of type '<type1>' cannot be converted to '<type2>'

Une instruction tente de convertir un type de données en un autre d'une façon non définie. Les causes de cette erreur peuvent être les suivantes :

- Une conversion spécifie deux types de données entre lesquels il n'existe aucune conversion. C'est le cas par exemple pour une conversion entre une valeur **Boolean** et le type **Date**.
- Une initialisation d'un tableau n'inclut pas d'accolades (**{}**) après une clause **New**. Dans ce cas, <type2> est au format 'tableau unidimensionnel de <type>'.

ID d'erreur : BC30311

Pour corriger cette erreur

- Vérifiez que l'expression peut être convertie dans le type de données de destination.
- Si <type2> est un tableau, assurez-vous que la clause **New** contient des parenthèses et des accolades après le nom du type. Le code suivant illustre l'initialisation correcte d'un tableau.

```
Dim anIntArray As Integer() = New Integer() {}
```

Voir aussi

Tâches

Comment : initialiser une variable tableau

Autres ressources

Conversions de type en Visual Basic

Une valeur de type '<type1>' ne peut pas être convertie en '<type2>', car '<type3>' n'est pas un type référence

Value of type '<type1>' cannot be converted to '<type2>' because '<type3>' is not a reference type

Une instruction tente de convertir un type tableau en un autre type tableau lorsque les types de données des éléments du tableau cible ne sont pas un type référence.

ID d'erreur : BC30333

Pour corriger cette erreur

- Vérifiez les types de données des éléments des deux tableaux afin de déterminer la source de l'erreur.

Voir aussi

Concepts

Conversion des tableaux

Autres ressources

Conversions de type en Visual Basic

Une valeur de type '<type1>' ne peut pas être convertie en '<type2>', car '<type3> n'est pas dérivé de '<type4>'

Value of type '<type1>' cannot be converted to '<type2>' because '<type3>' is not derived from '<type4>'

Une instruction tente de convertir un type tableau vers un autre type tableau lorsque les types de données des éléments des tableaux ne sont pas tous deux des types référence ou lorsqu'une conversion (restrictive ou étendue) est impossible entre les types d'éléments des deux tableaux.

ID d'erreur : BC30332

Pour corriger cette erreur

- Vérifiez les types de données des éléments des deux tableaux afin de déterminer la source de l'erreur.

Voir aussi

Concepts

Conversion des tableaux

Autres ressources

Tableaux dans Visual Basic

Une valeur de type '<type1>' ne peut pas être convertie en '<type2>', car les types tableau ont des nombres de dimensions différents

Value of type '<type1>' cannot be converted to '<type2>' because the array types have different numbers of dimensions

Vous avez tenté de convertir un tableau en un autre avec un nombre de dimensions différent.

ID d'erreur : BC30414

Pour corriger cette erreur

- Modifiez l'un ou les deux tableaux pour faire correspondre le nombre de dimensions.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

La variable '<NomVariable>' est passée par référence avant qu'une valeur ne lui ait été assignée

Message d'erreur
La variable '<NomVariable>' est passée par référence avant qu'une valeur ne lui ait été assignée. Une exception de référence null peut se produire au moment de l'exécution.
Variable '<variablename>' is passed by reference before it has been assigned a value. A null reference exception could result at run time.

Un appel de procédure passe une variable en tant qu'argument à un paramètre **ByRef** avant qu'une valeur soit assignée à la variable.

Si une valeur n'a jamais été assignée à une variable, elle stocke la valeur par défaut pour son type de données. Pour un type de données référence, cette valeur par défaut est Nothing (Visual Basic). La lecture d'une variable de référence qui a la valeur **Nothing** peut générer une exception NullReferenceException dans certains cas.

Le passage d'un argument à une procédure **ByRef** expose la variable sous-jacente de l'argument à des modifications possibles par la procédure.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42030

Pour corriger cette erreur

- Si la procédure doit assigner une valeur à la variable par l'intermédiaire de l'argument **ByRef**, et s'il importe peu que la variable contienne déjà une valeur, aucune mesure n'est nécessaire.
- Si la logique de la procédure lit l'argument avant de lui assigner une valeur, et si la variable est d'un type valeur, vérifiez que la logique de la procédure ne dépend pas du fait que la variable contienne ou non sa valeur par défaut.
- Si la logique de la procédure lit l'argument avant de lui assigner une valeur, et si la variable est d'un type référence, vérifiez que la logique de la procédure peut gérer la valeur **Nothing**. Par exemple, elle peut utiliser Try...Catch...Finally, instruction (Visual Basic) pour intercepter **NullReferenceException**.

Voir aussi

Tâches
Dépannage des variables en Visual Basic
Référence
Dim, instruction (Visual Basic)
ByRef
Concepts
Types valeur et types référence
Passage d'arguments par valeur et par référence
Déclaration de variable dans Visual Basic

La variable '<NomVariable>' est passée par référence avant qu'une valeur ne lui ait été assignée (variable de structure)

Message d'erreur
La variable '<NomVariable>' est passée par référence avant qu'une valeur ne lui ait été assignée Une exception de référence null peut se produire au moment de l'exécution. Assurez-vous que la structure ou que tous les membres de référence sont initialisés avant de les utiliser
Variable '<variablename>' is passed by reference before it has been assigned a value. A null reference exception could result at runtime. Make sure the structure or all the reference members are initialized before use

Un appel de procédure passe une variable de structure en tant qu'argument à un paramètre **ByRef** avant qu'une valeur soit assignée à la variable.

Si une valeur n'a jamais été assignée à une variable de structure, chaque membre de structure contient la valeur par défaut pour son type de données. Pour un type de données référence, cette valeur par défaut est Nothing (Visual Basic). La lecture d'un membre de référence qui a la valeur **Nothing** peut générer une exception NullReferenceException dans certains cas.

Le passage d'un argument à une procédure **ByRef** expose la variable sous-jacente de l'argument à des modifications possibles par la procédure.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42108

Pour corriger cette erreur

- Si la procédure doit assigner des valeurs à des membres de structure par l'intermédiaire de l'argument **ByRef**, et s'il importe peu que les membres contiennent déjà des valeurs, aucune mesure n'est nécessaire.
- Si la logique de la procédure lit un membre de structure avant de lui assigner une valeur, et si le membre est d'un type valeur, vérifiez que la logique de la procédure ne dépend pas du fait que le membre contienne ou non sa valeur par défaut.
- Si la logique de la procédure lit un membre de structure avant de lui assigner une valeur, et si le membre est d'un type référence, vérifiez que la logique de la procédure peut gérer la valeur **Nothing**. Par exemple, elle peut utiliser Try...Catch...Finally, instruction (Visual Basic) pour intercepter **NullReferenceException**.

Voir aussi

Tâches
Dépannage des variables en Visual Basic
Référence
Dim, instruction (Visual Basic)
ByRef
Structure, instruction
Concepts
Types valeur et types référence
Passage d'arguments par valeur et par référence
Déclaration de variable dans Visual Basic
Autres ressources
Structures constituées à partir de vos propres types de données

La variable '<NomVariable>' est utilisée avant qu'une valeur ne lui ait été assignée

Message d'erreur
La variable '<NomVariable>' est utilisée avant qu'une valeur ne lui ait été assignée. Une exception de référence null peut se produire au moment de l'exécution.
Variable '<variablename>' is used before it has been assigned a value. A null reference exception could result at run time.

Une application dispose d'au moins un chemin d'accès possible par l'intermédiaire de son code qui lit une variable avant qu'une valeur lui soit assignée.

Si une valeur n'a jamais été assignée à une variable, elle stocke la valeur par défaut pour son type de données. Pour un type de données référence, cette valeur par défaut est Nothing (Visual Basic). La lecture d'une variable de référence qui a la valeur **Nothing** peut générer une exception NullReferenceException dans certains cas.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42104

Pour corriger cette erreur

- Vérifiez votre logique de flux de contrôle et assurez-vous que la variable a une valeur valide avant que le contrôle passe à une instruction qui la lit.
- Pour garantir que la variable a toujours une valeur valide, initialisez-la dans le cadre de sa déclaration. Consultez « Initialisation » dans Dim, instruction (Visual Basic).

Voir aussi

Tâches
Dépannage des variables en Visual Basic
Référence
Dim, instruction (Visual Basic)
Concepts
Types valeur et types référence
Déclaration de variable dans Visual Basic

La variable '<nomvariable>'masque une variable dans un bloc englobant

Variable '<variablename>' hides a variable in an enclosing block

Une variable englobée dans un bloc possède le même nom qu'une autre variable locale.

ID d'erreur : BC30616

Pour corriger cette erreur

- Renommez la variable du bloc englobé avec un nom différent de toute autre variable locale. Par exemple :

```
Dim a, b, x As Integer
If a = b Then
   Dim y As Integer = 20 ' Uniquely named block variable.
End If
```

- Cette erreur vient généralement de l'utilisation de `Catch e As Exception` à l'intérieur d'un gestionnaire d'événements. Si tel est le cas, nommez la variable du bloc **Catch** `ex` plutôt que `e`.

- Cette erreur est également souvent due à la tentative d'accès à une variable locale déclarée à l'intérieur d'un bloc **Try** dans un bloc **Catch** séparé. Pour corriger cette erreur, déclarez la variable à l'extérieur de la structure **Try...Catch...Finally**.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic

La variable '<NomVariable>' est utilisée avant qu'une valeur ne lui ait été assignée (erreur Visual Basic)

Variable '<variablename>' is used before it has been assigned a value (Visual Basic Error)

Une variable est utilisée avant qu'une valeur lui ait été assignée, ce qui génère NullReferenceException.

ID d'erreur : BC42109

Pour corriger cette erreur

- Vérifiez qu'une valeur a été assignée à la variable.

Voir aussi

Référence

Dépannage des exceptions : System.NullReferenceException

Concepts

Déclaration de variable dans Visual Basic

Les instructions de déclaration de variable ne sont pas valides dans la fenêtre Exécution

Variable declaration statements are not valid in the Immediate window

Vous ne pouvez pas utiliser de déclarations de variable dans la fenêtre **Exécution**.

ID d'erreur : BC30709

Pour corriger cette erreur

- Ne déclarez pas de variables dans la fenêtre **Exécution**.

Voir aussi

Référence

Fenêtre Exécution

Autres ressources

Débogage dans Visual Studio

Déclaration de variable sans clause 'As' ; type d'Object pris par défaut

Variable declaration without an 'As' clause; type of Object assumed

Une déclaration de variable ne spécifie pas de clause **As**.

Une clause **As** identifie un type de données à associer à un élément de programmation. Dans Dim, instruction (Visual Basic), elle spécifie le type de données de la ou les variables. Si vous n'incluez pas une clause **As** dans l'instruction **Dim**, le type de données de la variable a la valeur **Object** par défaut.

Par défaut, ce message est un avertissement. Pour plus d'informations sur le masquage des avertissements ou le traitement des avertissements en tant qu'erreurs, consultez Configuration d'avertissements en Visual Basic.

ID d'erreur : BC42020

Pour corriger cette erreur

- Incluez une clause **As** dans l'instruction **Dim** pour spécifier le type de données de la variable.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

Déclaration de variable dans Visual Basic

Cette variable utilise un type Automation non pris en charge par Visual Basic

Variable uses an Automation type not supported in Visual Basic

Vous avez essayé d'utiliser une variable définie dans une bibliothèque de types ou d'objets dont un type de données n'est pas pris en charge par Visual Basic.

Pour corriger cette erreur

- Utilisez une variable d'un type reconnu par Visual Basic.

  - ou -

- Si vous rencontrez cette erreur lors de l'utilisation de **FileGet** ou **FileGetOBject**, vérifiez que le fichier que vous essayez d'utiliser a été écrit avec **FilePut** ou **FilePutObject**.

Voir aussi

Référence

Liste des types de données (Visual Basic)

Les variables des modules ne peuvent pas être déclarées '<spécificateur>'

Variables in Modules cannot be declared '<specifier>'

Vous avez utilisé un spécificateur (**MustInherit**, par exemple) sur une variable dans une instruction **Module**. Les modules ne peuvent jamais être instanciés, ne prennent pas en charge l'héritage et ne peuvent pas implémenter des interfaces.

ID d'erreur : BC30593

Pour corriger cette erreur

- Supprimez le spécificateur.

Voir aussi

Référence

Module, instruction

'Variant' n'est plus un type pris en charge ; utilisez le type 'Object' à la place

'Variant' is no longer a supported type; use the 'Object' type instead

Le type de données **Object** remplace **Variant**.

ID d'erreur : BC30804

Pour corriger cette erreur

- Déclarez l'élément en tant que **Object**.

Voir aussi

Référence

Object, type de données

Concepts

Modifications des types de données pour les utilisateurs de Visual Basic 6.0

Le compilateur Visual Basic 2005 n'est pas en mesure de récupérer à partir de l'erreur suivante : <erreur>

Visual Basic 2005 compiler is unable to recover from the following error: <error>

Le compilateur Visual Basic n'est pas en mesure de récupérer à partir de l'erreur suivante : <*error*>.

Certaines fonctionnalités IDE ont été désactivées.

Une erreur interne s'est produite dans le compilateur Visual Basic

ID d'erreur : BC31021

Pour corriger cette erreur

1. Enregistrez votre travail et redémarrez Visual Studio.
2. Si l'erreur se reproduit, réinstallez Visual Studio.
3. Si l'erreur persiste après la réinstallation, informez-en les services du support technique de Microsoft.

Voir aussi

Autres ressources

Support technique et accessibilité

Le numéro d'avertissement '<numéro>' pour l'option '<nomoption>' est non configurable ou non valide

Warning number '<number>' for the option '<optionname>' is either not configurable or not valid

Un ID d'avertissement n'est pas valide. Les ID doivent être des valeurs entières supérieures à 0.

ID d'erreur : BC2026

Pour corriger cette erreur

- Vérifiez que l'ID d'avertissement fourni est valide.

Voir aussi

Concepts

Configuration d'avertissements en Visual Basic

Avertissement considéré comme une erreur : <erreur>

Warning treated as error: <error>

Un avertissement a été rencontré lors de l'exécution d'un assembly compilé avec l'option **Considérer les avertissements comme des erreurs** activée.

ID d'erreur : BC31072

Pour corriger cette erreur

1. Corrigez l'avertissement.
2. Compilez l'assembly avec l'option **Considérer les avertissements comme des erreurs** désactivée.

Voir aussi

Référence

/warnaserror (Visual Basic)
Pages de propriétés, page Générer
Générer, Propriétés de configuration, boîte de dialogue Propriétés relatives aux projets de déploiement

Impossible d'évaluer des méthodes Web dans les fenêtres de débogage

Web methods cannot be evaluated in debug windows

Vous ne pouvez pas tester les méthodes du service Web XML dans la fenêtre de débogage.

ID d'erreur : BC30732

Pour corriger cette erreur

- Testez les méthodes du service Web XML à partir de l'éditeur de code.

Voir aussi

Autres ressources

Débogage dans Visual Studio

L'attribut 'WebMethod' n'aura aucun effet sur ce membre, car sa classe conteneur n'est pas exposée en tant que service Web

'WebMethod' attribute will not affect this member because its containing class is not exposed as a web service

L'attribut WebMethodAttribute autorise l'appel d'une méthode à partir des clients Web distants, mais uniquement lorsque la classe de la méthode dérive de WebService.

ID d'erreur : BC30771

Pour corriger cette erreur

- Modifiez la classe à dériver de **WebService**.

  - ou -

- Supprimez l'attribut **WebMethodAttribute** de la méthode.

Voir aussi

Tâches

Procédure pas à pas : création d'un service Web XML en utilisant Visual Basic ou Visual C#

Les instructions 'Wend' ne sont plus prises en charge ; utilisez les instructions 'End While' à la place

'Wend' statements are no longer supported; use 'End While' statements instead

La construction **End While** remplace le mot clé **Wend**.

ID d'erreur : BC30809

Pour corriger cette erreur

- Utilisez **End While** pour terminer un bloc **While**.

Voir aussi

Référence

While...End While, instruction (Visual Basic)
End (Visual Basic)

'While' doit se terminer par un 'End While' correspondant

'While' must end with a matching 'End While'

Une instruction **While** apparaît sans instruction **End While** correspondante. Une instruction **End While** doit être utilisée pour terminer le bloc **While**.

ID d'erreur : BC30082

Pour corriger cette erreur

1. Si ce bloc **While** fait partie d'un ensemble de blocs **While** imbriqués, vérifiez que chaque bloc est correctement terminé.
2. Ajoutez une instruction **End While** à la fin du bloc **While**.

Voir aussi

Référence

While...End While, instruction (Visual Basic)

'Widening' et 'Narrowing' ne peuvent pas être combinés

'Widening' and 'Narrowing' cannot be combined

Une instruction Operator spécifie Widening et Narrowing.

Lorsque vous définissez un opérateur de conversion, vous devez le déclarer comme **Widening** ou **Narrowing**. Ces caractéristiques étant mutuellement exclusives, vous ne pouvez pas spécifier les deux.

ID d'erreur : BC33001

Pour corriger cette erreur

- Supprimez le mot clé **Widening** ou **Narrowing** de l'instruction **Operator**. Vous devez spécifier l'un ou l'autre.

Voir aussi

Tâches

Comment : définir un opérateur

Comment : définir un opérateur de conversion

Référence

Operator, instruction

Concepts

Procédures d'opérateur

Les contextes et instructions 'With' ne sont pas valides dans les fenêtres de débogage

'With' contexts and statements are not valid in debug windows

Les instructions **With** ne peuvent être utilisées que dans du code source.

ID d'erreur : BC30726

Pour corriger cette erreur

- Supprimez les instructions **With** du code de débogage.

Voir aussi

Référence

With...End With, instruction (Visual Basic)

Autres ressources

Débogage dans Visual Studio

'With' doit se terminer par un 'End With' correspondant

'With' must end with a matching 'End With'

Une instruction **With** apparaît sans instruction **End With** correspondante. Une instruction **End With** doit être utilisée pour terminer le bloc **With**.

ID d'erreur : BC30085

Pour corriger cette erreur

- Si ce bloc **With** fait partie d'un ensemble de blocs **With** imbriqués, vérifiez que chaque bloc est correctement terminé.
- Ajoutez une instruction **End With** à la fin du bloc **With**.

Voir aussi

Référence

With...End With, instruction (Visual Basic)

La variable 'WithEvents' ne déclenche aucun événement

'WithEvents' variable does not raise any events

La variable objet déclarée avec le mot clé **WithEvents** ne fait pas référence à une instance de classe capable de déclencher des événements.

ID d'erreur : BC30591

Pour corriger cette erreur

- Vérifiez que la variable objet fait référence à une instance de classe capable de déclencher des événements.

Voir aussi

Concepts

Événements et gestionnaires d'événements

Événements et gestionnaires d'événements

La variable 'WithEvents' ne déclenche pas d'événements d'instance accessibles par '<nomconteneur>'

'WithEvents' variable does not raise any instance events that are accessible to '<containername>'

L'objet que vous référencez ne déclenche pas d'événements accessibles.

ID d'erreur : BC30600

Pour corriger cette erreur

- Modifiez les modificateurs d'accès de l'événement ou choisissez un événement accessible dans le contexte en cours.

Voir aussi

Concepts

Événements et gestionnaires d'événements

La longueur du nom de la variable WithEvents ne peut pas dépasser 1019 caractères

WithEvents variable name length cannot exceed 1019 characters

Le nom ou l'identificateur de chaque variable déclarée à l'aide de **WithEvents** est limité à 1019 caractères.

ID d'erreur : BC32205

Pour corriger cette erreur

- Réduisez la longueur de l'identificateur.

Voir aussi

Référence

WithEvents

Concepts

Noms d'éléments déclarés

Les variables 'WithEvents' peuvent seulement être de type classe, interface ou paramètre de type avec des contraintes de classe

'WithEvents' variables can only be typed as classes, interfaces or type parameters with class constraints

Vous avez déclaré une variable de type structure conjointement avec **WithEvents**, ce qui ne représente pas une utilisation valide du modificateur **WithEvents**.

ID d'erreur : BC30413

Pour corriger cette erreur

- Utilisez **AddHandler** pour gérer les événements définis dans une structure.

Voir aussi

Référence

Dim, instruction (Visual Basic)

AddHandler, instruction

Concepts

WithEvents et la clause Handles

Les variables 'WithEvents' ne peuvent pas être de type tableau

'WithEvents' variables cannot be typed as arrays

Vous avez tenté de déclarer un tableau en utilisant **WithEvents**. Vous pouvez déclarer le nombre de variables de votre choix à l'aide du mot clé **WithEvents**, mais vous ne pouvez pas déclarer de tableaux de cette manière.

ID d'erreur : BC30476

Pour corriger cette erreur

- Déclarez individuellement les variables.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

WithEvents et la clause Handles

Les variables 'WithEvents' doivent comporter une clause 'As'

'WithEvents' variables must have an 'As' clause

Vous n'avez pas fourni de clause **As** avec le mot clé **WithEvents**. Déclarez la variable comme la classe spécifique capable de déclencher les événements.

ID d'erreur : BC30412

Pour corriger cette erreur

- Ajoutez la clause **As** qui spécifie la classe capable de déclencher les événements.

Voir aussi

Référence

Dim, instruction (Visual Basic)

Concepts

WithEvents et la clause Handles

Les propriétés 'WriteOnly' ne peuvent pas avoir un modificateur d'accès pour 'Set'

'WriteOnly' properties cannot have an access modifier on 'Set'

Une déclaration de propriété **WriteOnly** spécifie les niveaux d'accès dans l'instruction Property et l'instruction Set (Visual Basic).

Vous pouvez toujours spécifier un niveau d'accès pour la propriété. Vous pouvez également spécifier un niveau d'accès différent pour au plus l'une de ses procédures de propriété (**Get** ou **Set**), pourvu qu'elle soit plus restrictive que le niveau d'accès de la propriété. Vous ne pouvez pas spécifier des niveaux d'accès pour les deux procédures de propriété.

ID d'erreur : BC31104

Pour corriger cette erreur

- Supprimez le modificateur d'accès de l'instruction **Set**. Il représente la propriété **WriteOnly** entière et vous ne pouvez pas avoir deux niveaux d'accès pour la propriété.

Voir aussi

Tâches

Comment : déclarer une propriété avec des niveaux d'accès mixtes

Concepts

Procédures Property

Une propriété 'WriteOnly' doit fournir un 'Set'

'WriteOnly' property must provide a 'Set'

Si une propriété est déclarée comme **WriteOnly**, elle doit fournir une procédure pour l'écriture de sa valeur.

ID d'erreur : BC30125

Pour corriger cette erreur

1. Veillez à inclure une procédure **Set** entre l'instruction **Property** et l'instruction **End Property**.
2. Vérifiez que les autres procédures à l'intérieur de la déclaration **Property** sont correctement terminées.

Voir aussi

Référence

Property, instruction
Set, instruction (Visual Basic)

Le nombre d'arguments de type passés à la méthode '<NomProcédureGénérique>' est incorrect

Wrong number of type arguments passed to method '<genericprocedurename>'

Une procédure générique est appelée avec un nombre d'arguments de type qui ne correspond pas au nombre de paramètres de type avec lequel elle est définie.

ID d'erreur : BC30951

Pour corriger cette erreur

- Fournissez un argument de type pour chaque paramètre de type défini pour la procédure générique.

  - ou -

- Appelez la procédure générique sans arguments de type, et laissez le compilateur déduire les arguments de type.

Voir aussi

Référence

AddressOf, opérateur

Liste de types

Concepts

Types génériques en Visual Basic

Procédures génériques dans Visual Basic

Erreur d'analyse de la documentation XML : la balise de début '<balise>' n'a pas de balise de fin correspondante

Message d'erreur
Erreur d'analyse de la documentation XML : la balise de début <balise> n'a pas de balise de fin correspondante. Le commentaire XML sera ignoré.
XML documentation parse error: Start tag <tag> doesn't have a matching end tag. XML comment will be ignored.

Le commentaire XML contient une balise de début, mais ne contient pas de balise de fin correspondante.

ID d'erreur : BC42316

Pour corriger cette erreur

- Ajoutez une balise de fin qui correspond à la balise de début.

  - ou -

- Si la balise ne contient aucun texte interne, tel que <seealso> (Visual Basic), spécifiez une barre oblique avant le signe "supérieur à".

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

Erreur d'analyse de la documentation XML : <erreur>

XML documentation parse error: <error>

Les messages suivants peuvent apparaître avec cette erreur. Pour plus d'informations, consultez les rubriques suivantes :

- La balise de commentaire XML '<balise>' s'affiche avec des attributs identiques plusieurs fois dans le même bloc de commentaires XML
- La balise de commentaire XML '<balise>' n'est pas autorisée pour un élément de langage '<élément>'
- Le paramètre de commentaire XML '<paramètre>' ne correspond pas à un paramètre pour l'instruction <motclé> correspondante
- Le paramètre de commentaire XML doit avoir un attribut 'name'
- Le commentaire XML a une balise avec un attribut 'cref' '<attribut>' qui n'a pas pu être résolu
- La balise de commentaire XML 'include' doit avoir un attribut '<attribut>'

ID d'erreur : BC42304

Voir aussi

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Concepts

Documentation de votre code avec le langage XML (Visual Basic)

Le paramètre de type de commentaire XML doit avoir un attribut 'name'

Message d'erreur
Le paramètre de type de commentaire XML doit avoir un attribut 'name'. Le commentaire XML sera ignoré.
XML comment type parameter must have a 'name' attribute. XML comment will be ignored.

La balise de commentaire XML requiert un attribut **name** qui manque ou incomplet.

ID d'erreur : BC42318

Pour corriger cette erreur

- Ajoutez ou complétez l'attribut **name** requis pour la balise.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

Le paramètre de type du commentaire XML <paramètre> ne correspond pas à un paramètre de type pour l'instruction <motclé> correspondante

Message d'erreur
Le paramètre de commentaire XML <paramètre> ne correspond pas à un paramètre pour l'instruction <motclé> correspondante. Le commentaire XML sera ignoré.
XML comment parameter <parameter> does not match a type parameter on the corresponding <keyword> statement. XML comment will be ignored.

Une balise de commentaire XML a été spécifiée pour une partie, mais pas la totalité, des paramètres de type dans une méthode.

ID d'erreur : BC42317

Pour corriger cette erreur

- Fournissez des balises de commentaire XML pour tous les paramètres de type dans la méthode.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

La balise de commentaire XML 'returns' n'est pas autorisée pour une propriété 'WriteOnly'

Message d'erreur
La balise de commentaire XML 'returns' n'est pas autorisée pour une propriété 'WriteOnly'. Le commentaire XML sera ignoré.
XML comment tag 'returns' is not permitted on a 'WriteOnly' Property. XML comment will be ignored.

Un commentaire XML pour une déclaration de propriété en écriture seule ne peut pas contenir une balise < returns >.

ID d'erreur : BC42313

Pour corriger cette erreur

- Supprimez la balise < returns > non prise en charge.

Voir aussi

Référence
<returns> (Visual Basic)
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Property, instruction
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

La balise de commentaire XML 'returns' n'est pas autorisée pour un élément de langage 'declare sub'

Message d'erreur
La balise de commentaire XML 'returns' n'est pas autorisée pour un élément de langage 'declare sub'. Le commentaire XML sera ignoré.
XML comment tag 'returns' is not permitted on a 'declare sub' language element. XML comment will be ignored.

Un commentaire XML pour une déclaration **Declare Sub** ne peut pas contenir de balise **<**returns**>**.

ID d'erreur : BC42315

Pour corriger cette erreur

- Supprimez la balise **<**returns**>** qui n'est pas prise en charge.

Voir aussi

Référence
<returns> (Visual Basic)
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Declare, instruction
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

La balise de commentaire XML 'include' doit avoir un attribut '<attribut>'

Message d'erreur
La balise de commentaire XML 'include' doit avoir un attribut <attribut>. Le commentaire XML sera ignoré.
XML comment tag 'include' must have a <attribute> attribute. XML comment will be ignored.

Un attribut **file** ou **path**, passé à la balise **<**include**>**, est manquant ou incomplet.

ID d'erreur : BC42310

Pour corriger cette erreur

- Ajoutez ou complétez l'attribut requis pour la balise <include>.

Voir aussi

Référence
<include> (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

La balise de commentaire XML '<balise>' n'est pas autorisée pour un élément de langage '<élément>'

Message d'erreur
La balise de commentaire XML <balise> n'est pas autorisée pour un élément de langage <élément>. Le commentaire XML sera ignoré.
XML comment tag <tag> is not permitted on a '<element>' element. XML comment will be ignored.

Une balise de commentaire XML n'est pas prise en charge par l'élément de langage auquel elle s'applique.

ID d'erreur : BC42306

Pour corriger cette erreur

- Supprimez la balise de commentaire XML non prise en charge.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

La balise de commentaire XML '<balise>' s'affiche avec des attributs identiques plusieurs fois dans le même bloc de commentaires XML

Message d'erreur
La balise de commentaire XML <balise> s'affiche avec des attributs identiques plusieurs fois dans le même bloc de commentaires XML. Le commentaire XML sera ignoré.
XML comment tag <tag> appears with identical attributes more than once in the same XML comment block. XML comment will be ignored.

Le commentaire XML contient plusieurs balises avec des attributs identiques.

ID d'erreur : BC42305

Pour corriger cette erreur

- Supprimez la balise dupliquée.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

Le paramètre de commentaire XML doit avoir un attribut 'name'

Message d'erreur
Le paramètre de commentaire XML doit avoir un attribut 'name'. Le commentaire XML sera ignoré.
XML comment parameter must have a 'name' attribute. XML comment will be ignored.

Un attribut **name** requis manque dans la balise XML.

ID d'erreur : BC42308

Pour corriger cette erreur

- Fournissez l'attribut **name** requis pour la balise.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

Le paramètre de commentaire XML '<paramètre>' ne correspond pas à un paramètre pour l'instruction <motclé> correspondante

Message d'erreur
Le paramètre de commentaire XML <paramètre> ne correspond pas à un paramètre pour l'instruction <motclé> correspondante. Le commentaire XML sera ignoré.
XML comment parameter <parameter> does not match a parameter on the corresponding <keyword> statement. XML comment will be ignored.

Une balise de commentaire XML a été spécifiée pour une partie, mais pas la totalité, des paramètres dans une méthode.

ID d'erreur : BC42307

Pour corriger cette erreur

- Fournissez la balise de commentaire XML pour tous les paramètres dans la méthode.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

Le commentaire XML doit être la première instruction sur une ligne

Message d'erreur
Le commentaire XML doit être la première instruction sur une ligne. Le commentaire XML sera ignoré.
XML comment must be the first statement on a line. XML comment will be ignored.

Un commentaire XML a été placé après un autre élément de langage, ce qui n'est pas autorisé.

ID d'erreur : BC42302

Pour corriger cette erreur

- Déplacez le commentaire XML sur une nouvelle ligne.

Voir aussi

Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)
Concepts
Documentation de votre code avec le langage XML (Visual Basic)

Le commentaire XML a une balise avec un attribut 'cref' '<attribut>' qui n'a pas pu être résolu

Message d'erreur
Le commentaire XML a une balise avec un attribut 'cref' <attribut> qui n'a pas pu être résolu. Le commentaire XML sera ignoré.
XML comment has a tag with a 'cref' attribute <attribute> that could not be resolved. XML comment will be ignored.

Les balises peuvent avoir un attribut **cref** qui désigne un lien vers un autre élément du XML en spécifiant le nom relatif de l'identificateur. Au moment de la compilation, le compilateur remplace la valeur par l'identificateur XML qualifié pour la valeur sur laquelle pointe l'utilisateur. Le compilateur utilise ses règles de résolution normales pour rechercher le type ou le membre.

ID d'erreur : BC42309

Pour corriger cette erreur

- Validez l'attribut **cref** afin qu'il pointe sur un élément de code valide.

Voir aussi

Tâches
Comment : créer une documentation XML en Visual Basic
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

L'exception de commentaire XML doit avoir un attribut 'cref'

XML comment exception must have a 'cref' attribute

La balise <exception> permet de documenter les exceptions qui peuvent être levées par une méthode. L'attribut **cref** requis désigne le nom d'un membre qui est vérifié par le générateur de documentation. Si le membre existe, il est traduit en nom d'élément canonique dans le fichier documentation.

ID d'erreur : BC42319

Pour corriger cette erreur

- Ajoutez l'attribut **cref** à l'exception comme suit :

```
'''<exception cref="member">description</exception>
```

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Référence

<exception> (Visual Basic)

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Le commentaire XML ne peut pas être appliqué plusieurs fois pour un <type> partiel

Message d'erreur
Le commentaire XML ne peut pas être appliqué plusieurs fois pour un <type> partiel. Les commentaires XML pour ce <type> seront ignorés.
XML comment cannot be applied more than once on a partial <type>. XML comments for this <type> will be ignored.

Un bloc de commentaires XML peut apparaître au-dessus d'une seule partie d'un type partiel.

Si un bloc de commentaires XML apparaît au-dessus de plusieurs parties d'un type partiel, cet avertissement est créé pour chaque bloc de commentaires, et le commentaire XML de niveau supérieur sera ignoré.

ID d'erreur : BC42314

Pour corriger cette erreur

- Supprimez les blocs de commentaires superflus.

Voir aussi

Tâches
Comment : créer une documentation XML en Visual Basic
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Un commentaire XML ne peut pas figurer dans une méthode ou une propriété

XML comment cannot appear within a method or a property

Un commentaire XML ne peut pas figurer dans une méthode ou une propriété. Le commentaire XML sera ignoré. Les blocs de commentaires XML ne peuvent pas apparaître dans des méthodes ou des propriétés.

ID d'erreur : BC42303

Pour corriger cette erreur

- Supprimez le bloc de commentaires ou déplacez-le vers un emplacement plus approprié.

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Le bloc de commentaires XML doit précéder immédiatement l'élément de langage auquel il se rapporte

Message d'erreur
Le bloc de commentaires XML doit précéder immédiatement l'élément de langage auquel il se rapporte. Le commentaire XML sera ignoré.
XML comment block must immediately precede the language element to which it applies. XML comment will be ignored.

Cette erreur se produit également si une balise est mal placée ou incorrecte.

ID d'erreur : BC42300

Pour corriger cette erreur

1. Déplacez le bloc de commentaires de façon à ce qu'il précède l'élément de langage auquel il s'applique. Assurez-vous que des caractères superflus n'ont pas été insérés par erreur avant la balise initiale.
2. Assurez-vous que la balise est valide.

Voir aussi

Tâches
Comment : créer une documentation XML en Visual Basic
Référence
Balises XML recommandées pour les commentaires de documentation (Visual Basic)

Le bloc de commentaires XML ne peut pas être associé à un élément de langage qui prend en charge l'application des commentaires de la documentation XML

XML comment block cannot be associated with any language element that supports the application of XML documentation comments

Le bloc de commentaires XML ne peut pas être associé à un élément de langage valide. Le commentaire XML sera ignoré. Cette erreur se produit également si une balise XML dans le bloc de commentaire n'a aucun élément de langage correspondant.

ID d'erreur : BC42312

Pour corriger cette erreur

- Recherchez et supprimez la balise XML superflue ou fournissez l'élément de langage qu'elle décrit.

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Référence

Balises XML recommandées pour les commentaires de documentation (Visual Basic)

l'option <option> ne peut pas être suivie par '+' ou '-' ; ignoré

option '<option>' cannot be followed by '+' or '-'; ignored

L'option du compilateur a été spécifiée avec la valeur **+** ou **-** ; ces valeurs ne sont pas prises en charge.

Pour plus d'informations sur les options du compilateur et les valeurs qu'elles prennent en charge, consultez Options du compilateur Visual Basic par ordre alphabétique.

ID d'erreur : BC2028

Pour corriger cette erreur et utiliser l'option

- Spécifiez l'option du compilateur sans valeur.

Pour corriger cette erreur et ne pas utiliser l'option

- Supprimez l'option du compilateur.

Voir aussi

Référence

Options du compilateur Visual Basic par ordre alphabétique

impossible d'envoyer le rapport d'erreur automatiquement

Message d'erreur
impossible d'envoyer le rapport d'erreur automatiquement. Visitez le site Web http://go.microsoft.com/fwlink/? LinkId=42039 pour configurer les paramètres d'envoi des rapports d'erreurs.
cannot send error report automatically. Please visit 'http://go.microsoft.com/fwlink/?LinkId=42039' to configure send error report settings.

Vous avez spécifié l'option de compilateur **/errorreport:send**, mais l'ordinateur n'est pas configuré pour envoyer automatiquement des rapports d'erreurs. Aucune information ne sera envoyée à propos des erreurs internes dans le compilateur Visual Basic.

ID d'erreur : BC2027

Pour corriger cette erreur

- Supprimez l'option de compilateur **/errorreport:send** ou remplacez-la par **/errorreport:queue**, **/errorreport:prompt**ou **/errorreport:none**.

  - ou -

- Activez le rapport d'erreurs automatique en suivant les instructions sur le site Web http://go.microsoft.com/fwlink/?LinkId=42039

Voir aussi

Référence
/errorreport
Autres ressources
http://go.microsoft.com/fwlink/?LinkId=42039

Impossible d'inclure un fragment XML '<CheminÉlément>' du fichier '<FichierCommentaires>'

Unable to include XML fragment '<elementpath>' of file '<commentfile>'

Une balise `<include>` dans un commentaire XML contient une expression XPath pour un élément qui n'existe pas ou le fichier contient du XML incorrect.

ID d'erreur : BC42320

Pour corriger cette erreur

1. Vérifiez que le XML dans le fichier est valide.
2. Vérifiez que le fragment XML existe dans le fichier.

Voir aussi

Référence

<include> (Visual Basic)

Code XML incorrect. Impossible d'inclure le fichier '<FichierCommentaires>'

Badly formed XML. File '<commentfile>' cannot be included

Une balise `<include>` dans l'un des commentaires XML fait référence à un fichier qui n'existe pas ou à un fichier qui contient du XML incorrect.

ID d'erreur : BC42321

Pour corriger cette erreur

1. Vérifiez que le fichier existe.
2. Vérifiez que le XML dans le fichier est valide.

Voir aussi

Référence

<include> (Visual Basic)

L'attribut 'StructLayout' ne peut pas être appliqué à un type générique

Attribute 'StructLayout' can not be applied to a generic type

Cette rubrique est nouvelle pour Visual Studio 2005 SP1.

Un type générique se voit appliquer l'attribut StructLayoutAttribute. Cette attribution n'est pas autorisée parce que l'attribut **StructLayoutAttribute** requiert que la structure soit connue au moment de la compilation, mais la structure d'un type générique peut changer après la compilation, car elle peut être construite pendant l'exécution avec plusieurs arguments de type différents.

ID d'erreur : BC30972

Pour corriger cette erreur

- Supprimez l'attribut **StructLayoutAttribute** du type générique.

Voir aussi

Concepts

Types génériques en Visual Basic
Vue d'ensemble des attributs dans Visual Basic

Une classe utilisée dans l'expression n'est pas chargée lors de la session de débogage

A class used in the expression is not loaded during debugging session

Cette rubrique est nouvelle pour Visual Studio 2005 SP1.

Lors du débogage, une classe a été référencée avant d'être chargée. Pour forcer le débogueur à charger la classe, créez une nouvelle instance de la classe et assignez-la à une variable temporaire. Vous pouvez ensuite réévaluer l'expression ayant provoqué l'erreur.

ID d'erreur : BC30973

Pour corriger cette erreur

1. Créez une instance de la classe en utilisant le mot clé **New**, puis assignez l'instance à une variable temporaire.
2. Évaluez l'expression qui a provoqué l'erreur initialement.

Voir aussi

Référence

New (Visual Basic)

Impossible d'évaluer les instructions de commentaires

Comment statements cannot be evaluated

Cette rubrique est nouvelle pour Visual Studio 2005 SP1.

Un commentaire a été utilisé dans la fenêtre de débogage.

ID d'erreur : BC30974

Pour corriger cette erreur

- N'utilisez pas de commentaires dans la fenêtre de débogage.

Voir aussi

Concepts

Commentaires dans le code

Autres ressources

Débogage dans Visual Studio

Variable de contrôle de boucle For '<nom_variable>' déjà utilisée par une boucle For englobante

For loop control variable already in use by an enclosing For loop

Cette rubrique est nouvelle pour Visual Studio 2005 SP1.

Les boucles **For** imbriquées doivent toutes utiliser des variables de contrôle uniques, également connues sous le nom *compteurs de boucle*.

ID d'erreur : BC30975

Pour corriger cette erreur

- Changez le nom de l'une des variables de contrôle.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

La variable de contrôle Next ne correspond pas à la variable de contrôle de boucle For '<nomvariable>'

Next control variable does not match For loop control variable

Cette rubrique est nouvelle pour Visual Studio 2005 SP1.

La variable de contrôle de l'instruction **Next** d'une boucle **For...Next** doit correspondre à la variable de l'instruction **For** correspondante.

ID d'erreur : BC30976

Pour corriger cette erreur

1. Vérifiez que l'orthographe de la variable dans l'instruction **Next** et celle de la variable dans l'instruction **For** correspondante sont identiques.
2. Vérifiez qu'aucune partie de la boucle englobante n'a été supprimée par inadvertance.
3. Si cette boucle fait partie d'un ensemble de boucles imbriquées, vérifiez que chaque boucle est correctement terminée.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

Les méthodes de 'System.Nullable(Of T)' ne peuvent pas être utilisées en tant qu'opérandes de l'opérateur 'AddressOf'

Methods of 'System.Nullable(Of T)' cannot be used as operands of the 'AddressOf' operator

Cette rubrique est nouvelle pour Visual Studio 2005 SP1.

Une instruction utilise l'opérateur **AddressOf** avec un opérande qui représente une procédure ou propriété de la structure Nullable.

ID d'erreur : BC32126

Pour corriger cette erreur

- Remplacez le nom de procédure ou de propriété dans la clause **AddressOf**. Vous devez utiliser un opérande qui ne soit pas membre de **Nullable**.

Voir aussi

Référence

AddressOf, opérateur

Nullable

Concepts

Types valeur dont la valeur n'est pas définie

Types génériques en Visual Basic

Messages d'exécution Visual Basic

Cette section contient les messages d'erreur Visual Basic qui se produisent au moment de l'exécution.

Voir aussi

Tâches

Comment : obtenir des informations sur les erreurs d'exécution Visual Basic

Autres ressources

Messages d'erreur (Visual Basic)

<adresse> n'est pas une adresse valide de fichier distant

Message d'erreur
<adresse> n'est pas une adresse valide de fichier distant. Une adresse valide doit inclure un protocole, un chemin d'accès et un nom de fichier.
<address> is not a valid remote file address. A valid address should include a protocol, a path and a filename.

Une adresse non valide a été spécifiée pour une opération de transfert ou de téléchargement. Les adresses doivent contenir un protocole, par exemple `ftp` ou `http`, un chemin d'accès valide et le nom de fichier. Cette erreur peut être due à une analyse incorrecte des chemins d'accès.

Pour corriger cette erreur

- Vérifiez l'adresse spécifiée pour déterminer la zone de défaillance, et entrez les informations nécessaires.

Voir aussi

Tâches
Comment : transférer un fichier dans Visual Basic
Comment : télécharger un fichier dans Visual Basic
Référence
My.Computer.Network.UploadFile, méthode
My.Computer.Network.DownloadFile, méthode
Autres ressources
Opérations de réseau dans le .NET Framework avec Visual Basic

Un délimiteur ne peut pas être Nothing ou un String vide

A delimiter cannot be Nothing or an empty String

TextFieldParser ne peut pas lire le fichier car la propriété **Delimiters** a la valeur **Nothing** ou est un **String** vide ("").

Pour corriger cette erreur

- Entrez une valeur valide pour **Delimiters**.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Référence

TextFieldParser.SetDelimiters, méthode

TextFieldParser.Delimiters, propriété

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Les guillemets doubles ne constituent pas un jeton de commentaire valide pour les champs délimités lorsque EscapeQuote a la valeur True

A double quote is not a valid comment token for delimited fields where EscapeQuote is set to True

Un guillemet a été fourni comme délimiteur pour **TextFieldParser**, mais **EscapeQuotes** a la valeur **True**.

Pour corriger cette erreur

- Affectez à **EscapeQuotes** la valeur **False**.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Référence

TextFieldParser.SetDelimiters, méthode
TextFieldParser.Delimiters, propriété
TextFieldParser, objet

Un journal a déjà été créé avec ce nom sur cet ordinateur

A log has already been created with this name on this machine

Une tentative a été effectuée pour créer un journal avec un nom déjà utilisé.

Pour corriger cette erreur

- Supprimez le journal en conflit.
- Utilisez un autre nom pour le journal.

Voir aussi

Tâches

Comment : écrire des messages de journal
Comment : enregistrer des messages lorsque l'application démarre ou s'arrête
Comment : écrire dans le journal des événements de l'application
Procédure pas à pas : détermination de l'emplacement des informations My.Application.Log
Procédure pas à pas : modification de l'emplacement des informations My.Application.Log

Référence

My.Application.Log, objet

Concepts

Utilisation des journaux d'applications dans Visual Basic

Un appel à une propriété ou une méthode ne peut pas utiliser une référence vers un objet privé, que ce soit comme argument ou comme valeur de retour

A property or method call cannot include a reference to a private object, either as an argument or as a return value

Cette erreur peut se produire pour de nombreuses raisons.

- Un client a appelé une propriété ou une méthode d'un composant out-of-process et a tenté de passer une référence à un objet privé comme l'un des arguments.
- Un composant out-of-process a appelé une méthode de rappel sur son client et a tenté de passer une référence à un objet privé.
- Un composant out-of-process a tenté de passer une référence à un objet privé comme un argument d'un événement qu'il déclenchait.
- Un client a tenté d'assigner une référence d'objet privé à un argument **ByRef** d'un événement qu'il gérait.

Pour corriger cette erreur

- Supprimez la référence.

Voir aussi

Référence

Private (Visual Basic)

Un formulaire de démarrage n'a pas été spécifié

A startup form has not been specified

L'application utilise la classe WindowsFormsApplicationBase, mais ne spécifie pas le formulaire de démarrage.

Cela peut se produire si la case à cocher **Activer l'infrastructure de l'application** est activée dans le concepteur de projets, mais que le **Formulaire de démarrage** n'a pas été spécifié. Pour plus d'informations, consultez Application, page du Concepteur de projets (Visual Basic).

Pour corriger cette erreur

1. Spécifiez un objet de démarrage pour l'application.

   Pour plus d'informations, consultez Comment : modifier l'objet de démarrage pour une application.

2. Substituez la méthode OnCreateMainForm pour affecter le formulaire de démarrage à la propriété MainForm.

Voir aussi

Référence

WindowsFormsApplicationBase
OnCreateMainForm
MainForm

Concepts

Vue d'ensemble du modèle d'application Visual Basic

Accès refusé à '<nom>'

Access denied to '<name>'

Un périphérique, fichier ou lecteur n'est pas accessible car les autorisations de sécurité de votre ordinateur ne sont pas suffisantes.

Pour corriger cette erreur

- Vérifiez que vous possédez les autorisations requises.

Voir aussi

Concepts

Autorisations de sécurité

L'ajout a échoué. Une valeur de clé dupliquée a été fournie

Add failed. Duplicate key value supplied

Une opération **Add** a échoué car la valeur de clé est identique à une autre valeur de clé. Les valeurs de clés doivent être uniques.

Pour corriger cette erreur

- Vérifiez que la valeur de clé est unique.

Voir aussi

Concepts

Types d'erreurs

Toutes les largeurs de champs, à l'exception de celle du dernier élément, doivent être supérieures à zéro

Message d'erreur
Toutes les largeurs de champs, à l'exception de celle du dernier élément, doivent être supérieures à zéro. Une largeur de champ inférieure ou égale à zéro dans le dernier élément indique que le dernier champ a une longueur variable.
All field widths, except the last element, must be greater than zero. A field width less than or equal to zero in the last element indicates the last field is of variable length.

L'opération a échoué car une largeur de champ autre que le dernier élément a une valeur inférieure ou égale à zéro. Une largeur de champ inférieure ou égale à zéro peut être utilisée comme dernier élément pour indiquer que le dernier champ a une longueur variable, mais elle ne peut pas être utilisée comme un autre élément.

Pour corriger cette erreur

- Affectez la valeur correcte à la largeur de champ.

Voir aussi

Tâches
Comment : lire des fichiers texte de largeur fixe dans Visual Basic
Référence
TextFieldParser.SetFieldWidths, méthode
TextFieldParser.FieldWidths, propriété
TextFieldParser, objet

Un nom incorrect a été spécifié pour le journal des événements

An invalid name was specified for the event log

Un nom incorrect a été spécifié pour le journal des événements. Cela est généralement dû à des caractères non valides dans le nom, à un nom de fichier vide ou à un nom de fichier trop long.

Pour corriger cette erreur

- Si le nom spécifié contient plus de huit caractères, vérifiez qu'il n'existe pas de conflit avec les noms d'autres journaux des événements. Seuls les huit premiers caractères sont pris en compte pour déterminer si le nom est unique.
- Si un chemin d'accès est fourni, vérifiez qu'il est correctement analysé.
- Vérifiez que le nom ne contient pas de caractères non valides. Les caractères qui ne peuvent pas être utilisés dans un nom de fichier sont : <, >, :, ", /, \ et |.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic
Comment : renommer un fichier dans Visual Basic
Comment : créer et supprimer des journaux des événements personnalisés

Une erreur inattendue s'est produite parce qu'une ressource de système d'exploitation requise pour le démarrage de l'instance unique ne peut pas être acquise

An unexpected error has occurred because an operating system resource required for single instance startup cannot be acquired

L'application ne peut pas acquérir une ressource de système d'exploitation requise. Ce problème peut avoir plusieurs causes :

- L'application ne dispose pas d'autorisations pour créer des objets de système d'exploitation nommés.
- Le Common Language Runtime ne dispose pas d'autorisations pour créer des fichiers mappés en mémoire.
- L'application doit accéder à un objet de système d'exploitation, mais un autre processus l'utilise.

Pour corriger cette erreur

1. Vérifiez que l'application dispose des autorisations nécessaires pour créer des objets de système d'exploitation nommés.
2. Vérifiez que le Common Language Runtime dispose des autorisations nécessaires pour créer des fichiers mappés en mémoire.
3. Redémarrez l'ordinateur pour effacer les processus qui utilisent la ressource requise pour se connecter à l'application de l'instance d'origine.
4. Prenez note des circonstances dans lesquelles l'erreur est survenue et appelez les services de Support Technique de Microsoft.

Voir aussi

Tâches

Comment : spécifier le comportement d'instanciation pour une application

Autres ressources

Présentation du débogueur

Support technique et accessibilité

Un autre journal des événements a déjà inscrit une source avec ce nom

Another event log has already registered a source with this name

Une tentative a été faite pour écrire une entrée dans un journal des événements où la source spécifiée est enregistrée dans un autre journal des événements.

Pour que votre composant puisse écrire une entrée dans un journal, vous devez d'abord définir la propriété Source de votre instance de composant EventLog. Le cas échéant, le système vérifie que la source que vous avez spécifiée est enregistrée dans le journal des événements dans lequel le composant écrit l'entrée et appelle CreateEventSource si nécessaire.

Pour corriger cette erreur

1. Supprimez l'association de la source au premier journal à l'aide de la méthode DeleteEventSource ou DeleteEventSource.
2. Enregistrez la source dans le nouveau journal.

Voir aussi

Tâches

Comment : supprimer une source d'événements

Comment : ajouter votre application en tant que source d'entrées d'un journal des événements

Référence

My.Application.Log, objet

Erreur définie par l'application ou par l'objet

Application-defined or object-defined error

Une erreur définie par l'application ou par l'utilisateur s'est produite. Vérifiez l'erreur.

Pour corriger cette erreur

- Consultez l'erreur pour découvrir l'emplacement où elle est définie et comment y remédier.

Voir aussi

Concepts

Types d'erreurs

L'argument '<argument1>' doit être inférieur ou égal à la longueur de l'argument '<argument2>'

Argument '<argument1>' must be less than or equal to the length of argument '<argument2>'

La longueur d'un argument dépasse la limite autorisée, déterminée par la longueur du second argument.

Pour corriger cette erreur

1. Vérifiez que la longueur de la valeur de l'argument se situe dans la plage valide.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées par ce calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<nomargument>' ne peut pas être un tableau multidimensionnel

Argument '<argumentname>' cannot be a multidimensional array

Le rang d'un argument de tableau est supérieur à 1, alors qu'il ne peut être qu'un tableau unidimensionnel.

Pour corriger cette erreur

- Remplacez le rang de l'argument par 1.

Voir aussi

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic

L'argument <NomArgument> ne peut pas être une chaîne vide ou avoir la valeur Nothing

Argument <argumentname> cannot be an empty string or Nothing

L'argument requiert une valeur autre qu'une chaîne vide ("") ou **Nothing**. Cela peut être dû à un calcul inexact de la valeur.

Pour corriger cette erreur

- Fournissez une valeur valide pour l'argument.

Voir aussi

Référence

Nothing (Visual Basic)

Dépannage des exceptions : System.NullReferenceException

Concepts

Types d'erreurs

L'argument '<nomargument>' ne peut pas être converti en une valeur numérique

Argument '<argumentname>' cannot be converted to a numeric value

Une conversion a tenté de convertir une variable, telle qu'une chaîne, en une valeur numérique.

Pour corriger cette erreur

- Recherchez une conversion significative.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

L'argument '<nomargument>' ne peut pas être converti en un type '<nomtype>'

Argument '<argumentname>' cannot be converted to type '<typename>'

Une conversion tente de modifier un argument en un type qui n'est pas valide.

Pour corriger cette erreur

- Remplacez le type par un type valide.

Voir aussi

Référence

Fonctions de conversion (Visual Basic)

L'argument '<nomargument>' ne peut pas être converti en un type 'Date'

Argument '<argumentname>' cannot be converted to type 'Date'

Une conversion a tenté de remplacer une variable en type **Date**, mais la valeur n'est pas reconnue comme une date valide.

Pour corriger cette erreur

- Recherchez une conversion significative.

Voir aussi

Référence

Type de données date (Visual Basic)

Autres ressources

Conversions de type en Visual Basic

L'argument '<NomArgument>' n'est pas une valeur valide

Argument '<argumentname>' is not a valid value

La valeur d'un argument n'est pas valide.

Pour corriger cette erreur

1. Vérifiez la valeur de l'argument.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées par ce calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' n'est pas valide pour le tableau

Argument '<argumentname>' is not valid for the array

Un tableau contient un argument qui n'est pas valide.

Pour corriger cette erreur

- Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

Autres ressources

Tableaux dans Visual Basic

L'argument '<NomArgument>' a la valeur Nothing

Argument '<argumentname>' is Nothing

Une expression contient un argument null.

Pour corriger cette erreur

1. Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' a la valeur Nothing ou est vide

Argument '<argumentname>' is Nothing or empty

Un nom d'argument n'est pas valide car il a la valeur **Nothing** ou est vide. Un argument doit avoir un type de données spécifique et un nom valide.

Pour corriger cette erreur

- Ajoutez un nom d'argument valide.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Application.Info.DirectoryPath, propriété

Nothing (Visual Basic)

L'argument '<nomargument>' doit être supérieur à 0 ou égal à -1

Argument '<argumentname>' must be greater than 0 or equal to -1

La valeur d'un argument est inférieure à 0, mais elle n'est pas égale à -1.

Pour corriger cette erreur

1. Vérifiez que la valeur de l'argument se situe dans la plage valide.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées par ce calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' doit être supérieur ou égal à 1

Argument '<argumentname>' must be greater than or equal to 1

La valeur d'un argument est inférieure à 1.

Pour corriger cette erreur

1. Vérifiez que la valeur de l'argument se situe dans la plage valide.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées par ce calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' doit être supérieur ou égal à -1

Argument '<argumentname>' must be greater than or equal to -1

La valeur d'un argument est inférieure à -1.

Pour corriger cette erreur

1. Vérifiez que la valeur de l'argument se situe dans la plage valide.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées par ce calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument <NomArgument> doit être supérieur ou égal à zéro

Argument <argumentname> must be greater than or equal to zero

Une valeur d'argument est inférieure à zéro. Les valeurs d'argument doivent être supérieures ou égales à zéro.

Pour corriger cette erreur

1. Assurez-vous que la valeur de l'argument est supérieure ou égale à zéro.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées lors du calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<nomargument>' doit être supérieur ou égal à zéro

Argument '<argumentname>' must be greater than or equal to zero

Un argument n'est pas valide, car il est inférieur à zéro.

Pour corriger cette erreur

1. Vérifiez l'orthographe des arguments. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' doit être supérieur à zéro

Argument '<argumentname>' must be greater than zero

Un argument n'est pas valide, car il est inférieur ou égal à zéro.

Pour corriger cette erreur

1. Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' doit être compris entre -32 768 et 65 535

Argument '<argumentname>' must be in the range of -32768 to 65535

La valeur d'un argument se situe en dehors de la plage autorisée comprise entre -32 768 et 65 535.

Pour corriger cette erreur

1. Vérifiez que la valeur de l'argument se situe dans la plage valide.
2. Si l'argument est le résultat d'un calcul, vérifiez les valeurs traitées par ce calcul.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

L'argument '<NomArgument>' doit être compris entre 0 et 99

Argument '<argumentname>' must be within the range 0 to 99

Un argument n'est pas valide car il se situe en dehors de la plage comprise entre 0 et 99.

Pour corriger cette erreur

1. Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

L'argument '<nomargument>' doit être compris entre 1 et 255

Argument '<argumentname>' must be within the range 1 to 255

Un argument n'est pas valide car il se situe en dehors de la plage comprise entre 0 et 255.

Pour corriger cette erreur

1. Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

L'argument '<nomargument1>' doit être inférieur ou égal à la longueur de l'argument '<nomargument2>'

Argument '<argumentname1>' must be less than or equal the length of argument '<argumentname2>'

La longueur d'un argument fourni dépasse la longueur de l'argument qu'il remplace.

Pour corriger cette erreur

- Attribuez au premier argument une longueur inférieure ou égale à la longueur du second argument.

Voir aussi

Concepts

Types d'erreurs

L'argument 'Access' n'est pas valide. Les valeurs autorisées pour le mode Append sont 'OpenAccess.Write' et 'OpenAccess.Default'.

Argument 'Access' is not valid. Valid values for Append mode are 'OpenAccess.Write' and 'OpenAccess.Default'

Une valeur d'accès n'est pas valide en mode Append.

Pour corriger cette erreur

- Remplacez la valeur par **OpenAccess.Write** ou **OpenAccess.Default**.

Voir aussi

Référence

OpenAccess, énumération

L'argument 'Access' n'est pas valide. Les valeurs autorisées pour le mode Input sont 'OpenAccess.Read' et 'OpenAccess.Default'.

Argument 'Access' is not valid. Valid values for Input mode are 'OpenAccess.Read' and 'OpenAccess.Default'

Une valeur d'accès n'est pas valide en mode **Input**.

Pour corriger cette erreur

- Remplacez la valeur par **OpenAccess.Read** ou **OpenAccess.Default**.

Voir aussi

Référence

OpenAccess, énumération

L'argument 'Access' n'est pas valide. Les valeurs autorisées pour le mode Output sont 'OpenAccess.Write' et 'OpenAccess.Default'.

Argument 'Access' is not valid. Valid values for Output mode are 'OpenAccess.Write' and 'OpenAccess.Default'

Une valeur d'accès n'est pas valide en mode Output.

Pour corriger cette erreur

- Remplacez la valeur par **OpenAccess.Write** ou **OpenAccess.Default**.

Voir aussi

Référence

OpenAccess, énumération

L'argument BasePath doit être un chemin d'accès à un dossier

Argument BasePath must be a path to a folder

L'argument *BasePath* doit contenir un chemin d'accès à un dossier. Il est possible qu'une chaîne n'ait pas été correctement analysée et qu'une valeur qui n'est pas reconnue comme un chemin d'accès valide ait été fournie.

Pour corriger cette erreur

- Vérifiez que la valeur fournie pour *BasePath* correspond à un chemin d'accès valide à un dossier.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

BasePath

BasePath

BasePath

L'argument 'Life' ne peut pas être égal à zéro

Argument 'Life' cannot be zero

Un argument pour *Life*, qui doit être un type **Double** indiquant la durée du cycle de vie du bien, n'est pas valide.

Pour corriger cette erreur

- Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
- Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Référence

DDB, fonction
SYD, fonction
SLN, fonction

Concepts

Passage d'arguments par valeur et par référence

L'argument 'Conversion' n'est pas valide

Argument 'Conversion' is not valid

Une valeur de l'énumération **VbStrConv** n'est pas valide, probablement car elle est inférieure à zéro. Les valeurs d'énumération doivent être supérieures ou égales à zéro.

Pour corriger cette erreur

- Assurez-vous que la valeur est supérieure ou égale à zéro.

Voir aussi

Référence

VbStrConv, énumération

L'argument ne peut pas avoir la valeur Nothing

Argument cannot be Nothing

Une valeur null a été fournie pour un argument qui doit avoir une valeur.

Pour corriger cette erreur

- Vous avez peut-être tenté d'utiliser un objet sans fournir une instance de celui-ci. Dans ce cas, utilisez le mot clé **New** pour créer l'instance.
- Vérifiez que la valeur est correctement calculée.

Voir aussi

Référence

Dépannage des exceptions : System.NullReferenceException

L'argument ne peut pas être inférieur à zéro

Argument cannot be less than zero

Une valeur inférieure à zéro a été fournie, mais l'argument ne peut pas être inférieur à zéro.

Pour corriger cette erreur

- Vérifiez comment la valeur en question est calculée pour vous assurer qu'elle est correcte.

Voir aussi

Autres ressources

Présentation du débogueur

L'argument ne peut être une chaîne vide

Argument cannot be an empty string

Un argument a été défini en tant que chaîne vide ("").

Pour corriger cette erreur

- Fournissez une valeur valide pour l'argument.

Voir aussi

Concepts

Types d'erreurs

Argument non facultatif (Visual Basic)

Argument not optional (Visual Basic)

Le nombre et les types d'arguments doivent correspondre à ceux qui sont attendus. Soit le nombre d'arguments est incorrect, soit un argument omis n'est pas facultatif. Un argument ne peut être omis de l'appel à une procédure définie par l'utilisateur que s'il a été déclaré **Optional** dans la définition de procédure.

Pour corriger cette erreur

1. Fournissez tous les arguments nécessaires.
2. Assurez-vous que les arguments omis sont facultatifs. S'ils ne le sont pas, fournissez l'argument dans l'appel ou déclarez le paramètre **Optional** dans la définition.

Voir aussi

Concepts

Types d'erreurs

Les arguments ne sont pas valides

Arguments are not valid

Des arguments non valides sont passés à une fonction ou à une sous-routine.

Pour corriger cette erreur

1. Vérifiez les arguments afin de déterminer la raison pour laquelle ils ne sont pas valides.
2. Si les arguments sont les résultats de calculs, vérifiez les valeurs traitées par ces calculs.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

La valeur d'argument '<NomCheminAccès>' contient des caractères qui ne sont pas valides dans un nom de chemin d'accès

Argument value '<pathname>' contains characters that are not valid in a path name

Un chemin d'accès contient au moins un caractère qui n'est pas valide.

Pour corriger cette erreur

- Supprimez le caractère du nom du chemin d'accès.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Application.Info.DirectoryPath, propriété

L'argument 'Period' doit être inférieur ou égal à l'argument 'Life'

Argument 'Period' must be less than or equal to argument 'Life'

La valeur de l'argument *Period*, qui indique la période pour le calcul de l'amortissement du bien, est supérieure à la valeur de l'argument *Life*.

Pour corriger cette erreur

- Vérifiez que les arguments *Life* et *Period* sont exprimés dans les mêmes unités. Par exemple, si l'argument *Life* est mesuré en mois, il doit en être de même pour l'argument *Period*.

Voir aussi

Référence

DDB, fonction

SYD, fonction

Concepts

Passage d'arguments par valeur et par référence

L'argument 'Per' n'est pas valide

Argument 'Per' is not valid

Un argument fourni pour *Per*, qui doit être de type **Double** indiquant la périodicité des paiements dans la plage comprise entre 1 et la valeur *NPer*, n'est pas valide.

Pour corriger cette erreur

- Vérifiez que l'argument se situe dans la plage valide.

Voir aussi

Référence

IPmt, fonction

PPmt, fonction

Concepts

Passage d'arguments par valeur et par référence

L'argument 'Path' a la valeur Nothing ou est vide

Argument 'Path' is Nothing or empty

Un nom de chemin d'accès n'est pas valide car il a la valeur **Nothing** ou il est vide. Un chemin d'accès doit avoir un type de données spécifique et un nom valide.

Pour corriger cette erreur

- Ajoutez un nom de chemin d'accès valide.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Application.Info.DirectoryPath, propriété

Nothing (Visual Basic)

L'argument 'NPer' doit être supérieur à zéro

Argument 'NPer' must be greater than zero

La fonction **NPer**, qui retourne une valeur de type **Double** indiquant le nombre d'échéances d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe, nécessite un argument supérieur à zéro.

Pour corriger cette erreur

- Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
- Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Référence

NPer, fonction

Concepts

Passage d'arguments par valeur et par référence

Les dimensions du tableau ne correspondent pas à celles spécifiées par l'attribut 'VBFixedArray'

Array dimensions do not match those specified in the 'VBFixedArray' attribute

Les dimensions de votre tableau ne correspondent pas à celles définies dans l'attribut **VBFixedArray**.

Pour corriger cette erreur

- Faites correspondre les dimensions du tableau dans l'instruction **Dim** du tableau à celles définies par l'attribut **VBFixedArray**.

Voir aussi

Référence

ReDim, instruction (Visual Basic)

VBFixedArrayAttribute, classe

Concepts

Vue d'ensemble des tableaux dans Visual Basic

L'objet Automation n'a pas de valeur par défaut

Automation object does not have a default value

Visual Basic ne peut pas déterminer le membre par défaut de l'objet spécifié.

Pour corriger cette erreur

- Vérifiez la documentation de l'objet et précisez une spécification explicite pour la propriété ou méthode.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Erreur Automation

Automation error

Une erreur s'est produite pendant l'exécution d'une méthode ou l'obtention ou le paramétrage de la propriété d'une variable objet. L'erreur a été signalée par l'application qui a créé l'objet.

Pour corriger cette erreur

1. Vérifiez les propriétés de l'objet **Err** afin de déterminer l'origine et la nature de l'erreur.
2. Utilisez l'instruction **On Error Resume Next** immédiatement avant l'instruction d'accès, puis recherchez la présence d'erreurs immédiatement après l'instruction d'accès.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Convention d'appel de DLL incorrecte

Bad DLL calling convention

Les arguments passés à une bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) doivent correspondre exactement à ceux qui sont attendus par la routine. L'appel de conventions traite des nombres, types et ordres des arguments. Votre programme peut appeler une routine dans une DLL à laquelle est passé un type ou un nombre incorrect d'arguments.

Pour corriger cette erreur

1. Vérifiez que tous les types d'argument correspondent à ceux spécifiés dans la déclaration de la routine que vous appelez.
2. Vérifiez que vous passez le même nombre d'arguments que celui qui est indiqué dans la déclaration de la routine que vous appelez.
3. Si la routine DLL attend des arguments par valeur, vérifiez que **ByVal** est spécifié pour ces arguments dans la déclaration de la routine.

Voir aussi

Référence

Call, instruction (Visual Basic)
Declare, instruction

Concepts

Types d'erreurs

Mode de fichier incorrect

Bad file mode

Les instructions utilisées dans la manipulation du contenu de fichier doivent être appropriées au mode dans lequel le fichier a été ouvert. Plusieurs causes sont possibles :

- Une instruction **FilePutObject** ou **FileGetObject** spécifie un fichier séquentiel.
- Une instruction **Print** spécifie un fichier ouvert pour un mode d'accès autre que **Output** ou **Append**.
- Une instruction **Input** spécifie un fichier ouvert pour un mode d'accès autre que **Input**.
- Tentative d'écriture sur une propriété en lecture seule.

Pour corriger cette erreur

- Vérifiez que **FilePutObject** et **FileGetObject** font uniquement référence à des fichiers ouverts pour un accès **Random** ou **Binary**.
- Vérifiez que **Print** spécifie un fichier ouvert pour un mode d'accès **Output** ou **Append**. Si ce n'est pas le cas, utilisez une instruction différente pour placer les données dans le fichier ou rouvrez le fichier dans un mode approprié.
- Vérifiez que **Input** spécifie un fichier ouvert pour un mode d'accès **Input**. Si tel n'est pas le cas, utilisez une instruction différente pour placer les données dans le fichier ou rouvrez le fichier dans un mode approprié.
- Si vous écrivez dans un fichier en lecture seule, modifiez l'état de lecture/écriture du fichier ou n'essayez pas d'écrire dans celui-ci.
- Utilisez les fonctionnalités disponibles dans l'objet **My.Computer.FileSystem**.

Voir aussi

Tâches

Dépannage : lecture et écriture dans des fichiers texte

Référence

My.Computer.FileSystem, objet

Nom ou numéro de fichier incorrect

Bad file name or number

Une erreur s'est produite lors de la tentative d'accès au fichier spécifié. Cette erreur peut se produire pour de nombreuses raisons.

- Une instruction fait référence à un fichier par un nom de fichier ou un nombre non spécifié dans l'instruction **FileOpen** ou qui a été spécifié dans une instruction **FileOpen**, mais qui a été fermé ultérieurement.
- Une instruction fait référence à un fichier par un nombre situé en dehors de la plage des nombres de fichier.
- Une instruction fait référence à un nom ou nombre de fichier qui n'est pas valide.

Pour corriger cette erreur

1. Vérifiez que le nom de fichier est spécifié dans une instruction **FileOpen**. Notez que si vous avez appelé l'instruction **FileClose** sans arguments, vous avez peut-être fermé tous les fichiers ouverts par inadvertance.
2. Si votre code génère des nombres de fichier par algorithme, vérifiez que les nombres sont valides.
3. Vérifiez que les noms de fichiers sont conformes aux conventions du système d'exploitation.

Voir aussi

Référence

FileOpen, fonction

Concepts

Conventions d'affectation de noms Visual Basic

Longueur d'enregistrement incorrecte

Bad record length

Cette erreur peut se produire pour de nombreuses raisons.

- La longueur d'une variable d'enregistrement spécifiée dans une instruction **FileGet**, **FileGetObject**, **FilePut** ou **FilePutObject** diffère de la longueur spécifiée dans l'instruction **FileOpen** correspondante.
- La variable d'une instruction **FilePut** ou **FilePutObject** est ou inclut une chaîne à longueur variable.
- La variable d'une instruction **FilePut** ou **FilePutObject** est ou inclut un type **Variant.**

Pour corriger cette erreur

1. Vérifiez que la somme des tailles des variables de longueur fixe dans le type défini par l'utilisateur qui définit le type de la variable d'enregistrement est identique à la valeur indiquée dans la clause *Len* de l'instruction **FileOpen**.
2. Si la variable d'une instruction **FilePut** ou **FilePutObject** est ou inclut une chaîne de longueur variable, vérifiez que cette dernière possède au moins 2 caractères de moins que la longueur d'enregistrement spécifiée dans la clause *Len* de l'instruction **FileOpen**.
3. Si la variable d'une instruction **FilePut** ou **FilePutObject** est ou inclut un type **Variant**, vérifiez que la chaîne de longueur variable possède au moins 4 octets de moins que la longueur d'enregistrement spécifiée dans la clause *Len* de l'instruction **FileOpen**.

Voir aussi

Référence

FileGet, fonction
FileGetObject, fonction
FilePut, fonction
FilePutObject, fonction

Numéro d'enregistrement incorrect

Bad record number

Le numéro d'enregistrement dans une instruction **a FileGet**, **FilePut**, **FileGetObject** ou **FilePutObject** est inférieur ou égal à zéro.

Pour corriger cette erreur

- Vérifiez les calculs utilisés pour générer le numéro d'enregistrement. Vérifiez l'orthographe des variables contenant le numéro d'enregistrement ou utilisées dans le calcul des numéros d'enregistrement. Un nom de variable mal orthographié est déclaré implicitement et possède une valeur égale à zéro, sauf si vous avez utilisé **Option Explicit On** dans le module.

Voir aussi

Référence

Option Explicit, instruction (Visual Basic)

BaseLogName ne peut pas être Nothing ou un String vide

BaseLogName cannot be Nothing or an empty String

La valeur de la propriété BaseFileName ne peut pas être **Nothing** ou une chaîne vide.

La propriété **BaseFileName** spécifie le nom de base des fichiers journaux.

Pour corriger cette erreur

- Affectez à la propriété **BaseFileName** une chaîne qui contient au moins un caractère.

Voir aussi

Référence

My.Application.Log, objet
My.Log, objet
BaseFileName

BaudRate doit être supérieur à 0.

BaudRate must be greater than 0

L'argument *BaudRate* spécifié pour la méthode **My.Computer.Ports.OpenSerialPort** doit être supérieur à zéro.

Pour corriger cette erreur

- Remplacez la valeur de l'argument *BaudRate* par une valeur positive.

Voir aussi

Référence

My.Computer.Ports.OpenSerialPort, méthode

Impossible de combiner les arguments 'Before' et 'After'

'Before' and 'After' arguments cannot be combined

Une liste d'arguments inclut les arguments *Before* et *After*, qui s'excluent mutuellement.

Pour corriger cette erreur

- Supprimez l'argument *Before* ou *After*.

Voir aussi

Référence

XmlNodeOrder Enumeration

Impossible de convertir la valeur de départ de type '<type1>', la valeur de limite de '<type2>' et la valeur de pas de '<type 3>' en type régulier

Cannot convert start value of type '<type1>', limit value of '<type2>' and step value of '<type3>' to a common type

Les valeurs de contrôle **For...Next** spécifiées ont des types différents et ne sont donc pas valides car elles ne peuvent pas être converties en un type régulier.

Pour corriger cette erreur

- Remplacez le type d'une ou plusieurs valeurs afin qu'elles puissent toutes être converties en un type régulier.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

Concepts

Débogage pour les utilisateurs de Visual Basic 6.0

Impossible de convertir la valeur de départ de type '<type1>' et la valeur de pas de type '<type2>' en type régulier

Cannot convert start value of type '<type1>' and step value of type '<type2>' to a common type

Les valeurs de contrôle **For...Next** spécifiées ont des types différents et ne sont donc pas valides car elles ne peuvent pas être converties en un type régulier.

Pour corriger cette erreur

- Remplacez le type d'une ou plusieurs valeurs afin qu'elles puissent toutes être converties en un type régulier.

Voir aussi

Référence

For...Next, instruction (Visual Basic)

Concepts

Contrôle de l'exécution

Débogage pour les utilisateurs de Visual Basic 6.0

Impossible de convertir l'argument '<nomargument>' de type '<type1>' en type '<type2>'

Cannot convert argument '<argumentname>' of type '<type1>' to type '<type2>'

Une opération de conversion tente de convertir un argument en un type incompatible.

Pour corriger cette erreur

- Vérifiez que le type dans lequel vous tentez de convertir l'argument est valide.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

Impossible d'appeler une fonction Friend pour un objet qui n'est pas une instance de la classe de définition

Cannot call friend function on object which is not an instance of defining class

Vous avez tenté d'appeler la procédure **Friend** d'une classe ou vous avez tenté d'accéder à une propriété ou méthode **Friend** interprocessus ou inter-threads. Une procédure **Friend** peut être appelée à partir d'un module situé en dehors de la classe, mais fait partie du projet dans lequel la classe est définie.

Pour corriger cette erreur

- Vérifiez que vous appelez ou que vous accédez à la procédure à partir d'un module qui fait partie du projet dans lequel la classe est définie.

Voir aussi

Référence

Friend (Visual Basic)

Impossible de calculer le taux à l'aide des arguments fournis

Cannot calculate rate using the arguments provided

Le calcul du taux ne contient pas les arguments requis.

Pour corriger cette erreur

1. Vérifiez l'orthographe des arguments dans l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Référence

Liste des mots clés financiers

Impossible de calculer le nombre de périodes à l'aide des arguments fournis

Cannot calculate number of periods using the arguments provided

Un appel à la fonction **NPer** ne contient pas tous les arguments requis.

Pour corriger cette erreur

- Vérifiez que les valeurs *Rate*, *Prnt* et *PV* sont incluses dans l'appel de fonction.

Voir aussi

Référence

NPer, fonction

Impossible de créer le composant ActiveX

Cannot create ActiveX Component

Vous avez tenté de placer un contrôle ActiveX sur un formulaire au moment du design ou d'ajouter un formulaire à un projet avec un contrôle ActiveX, mais les informations associées dans le Registre sont introuvables.

Pour corriger cette erreur

- Les informations du Registre ont peut-être été supprimées ou endommagées. Réinstallez le contrôle ActiveX ou contactez le fournisseur du contrôle.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Impossible de supprimer une ruche du Registre

Cannot delete a registry hive

Une tentative a été effectuée pour supprimer une ruche de Registre. Les ruches sont des clés de Registre de niveau supérieur, telles que **HKEY_CURRENT_USER** et **HKEY_LOCAL_MACHINE**, qui ne peuvent pas être supprimées.

Pour corriger cette erreur

- Vérifiez la clé de Registre que vous essayez de supprimer, et assurez-vous que le chemin d'accès correct est fourni.

Voir aussi

Tâches

Comment : supprimer une clé de Registre dans Visual Basic

Référence

My.Computer.Registry, objet

Concepts

Tâches courantes relatives au Registre

Impossible de déterminer le type du tableau, car sa valeur est Nothing

Cannot determine array type because it is Nothing

Le type de votre tableau ne peut pas être déterminé, car sa valeur est **Nothing**.

Pour corriger cette erreur

- Indiquez une valeur autre que **Nothing** pour le tableau.

Voir aussi

Référence

Nothing (Visual Basic)

Autres ressources

Tableaux dans Visual Basic

Impossible de renommer avec un lecteur différent

Cannot rename with different drive

L'instruction **Rename** ne peut pas spécifier de lettre de lecteur dans le chemin d'accès au fichier.

Pour corriger cette erreur

- Supprimez la lettre de lecteur du chemin d'accès au fichier.

Voir aussi

Référence

Rename, fonction

Impossible d'enregistrer le fichier dans TEMP

Cannot save file to TEMP

Soit un composant ne peut pas trouver un répertoire nommé TEMP, soit l'espace disponible sur un lecteur ou une partition contenant le répertoire TEMP est insuffisant pour enregistrer les informations.

Pour corriger cette erreur

1. Créez un répertoire nommé « TEMP » et définissez la variable d'environnement TEMP comme son chemin d'accès.
2. Faites de la place sur le lecteur en supprimant des fichiers inutiles ou créez un répertoire TEMP sur une autre partition et définissez la variable d'environnement TEMP comme son chemin d'accès.

Voir aussi

Concepts

Types d'erreurs

Impossible de créer le fichier temporaire nécessaire

Can't create necessary temporary file

Le lecteur contenant le répertoire spécifié par la variable d'environnement TEMP est plein ou la variable d'environnement TEMP spécifie un lecteur ou un répertoire non valide ou en lecture seule.

Pour corriger cette erreur

1. Supprimez des fichiers du lecteur, si celui-ci est plein.
2. Spécifiez un lecteur différent dans la variable d'environnement TEMP.
3. Spécifiez un lecteur valide pour la variable d'environnement TEMP.
4. Supprimez la restriction en lecture seule du lecteur ou répertoire actuellement spécifié.

Voir aussi

Concepts

Types d'erreurs

Impossible d'effectuer l'opération demandée (Visual Basic)

Can't perform requested operation (Visual Basic)

Une opération ne peut pas être effectuée car elle invaliderait l'état actuel du projet. Par exemple, vous avez peut-être essayé de modifier par programme le code en cours d'exécution.

Pour corriger cette erreur

- Arrêtez l'exécution du code, apportez les modifications le cas échéant, puis recommencez l'opération.

Voir aussi

Concepts

Types d'erreurs

Cast de la chaîne "<chaîne>" en type '<nomtype>' non valide

Cast from string "<string>" to type '<typename>' is not valid

Une tentative de convertir une variable chaîne en un autre type n'est pas valide.

Pour corriger cette erreur

- Vérifiez les types impliqués dans la conversion et assurez-vous que vous tentez une conversion en un type significatif.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

Cast du type '<nomtype1>' en type '<nomtype2>' non valide

Cast from type '<typename1>' to type '<typename2>' is not valid

Une tentative de convertir une variable d'un type en un autre n'est pas valide.

Pour corriger cette erreur

- Vérifiez les types impliqués dans la conversion et assurez-vous que vous tentez une conversion en un type significatif.

Voir aussi

Autres ressources

Conversions de type en Visual Basic

La classe '<nomclasse>' n'implémente pas l'interface System.Collections.ICollection

Class '<classname>' does not implement the System.Collections.ICollection interface

La classe n'implémente pas l'interface **System.Collections.ICollection**, qui définit la taille, les énumérateurs et les méthodes de synchronisation de toutes les collections. Vous devrez peut-être réinstaller Visual Studio.

Pour corriger cette erreur

- Réinstallez Visual Studio.

Voir aussi

Référence

ICollection Interface

La classe ne prend pas en charge Automation ou l'interface attendue

Class does not support Automation or does not support expected interface

Soit la classe spécifiée dans l'appel de fonction **GetObject** ou **CreateObject** n'a pas exposé d'interface de programmabilité, soit vous avez modifié un projet de .dll en .exe, ou inversement.

Pour corriger cette erreur

1. Vérifiez la documentation de l'application qui a créé l'objet pour connaître les limites d"utilisation d'automation avec cette classe d'objet.
2. Si vous avez modifié un projet .dll en .exe (ou inversement), vous devez annuler manuellement l'inscription de l'ancien .dll ou .exe.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

La classe n'est pas inscrite sur l'ordinateur local

Class not registered on local machine

Cette erreur provient généralement d'un fichier endommagé. Vous devrez peut-être réinstaller Visual Studio.

Pour corriger cette erreur

- Réinstallez Visual Studio.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Ressource de code introuvable

Code resource not found

Un appel a été effectué à une procédure dans une ressource de code, mais cette dernière est introuvable.

Pour corriger cette erreur

- Vérifiez que la ressource est disponible et correctement référencée.

Voir aussi

Concepts

Types d'erreurs

Erreur de verrouillage de ressource de code

Code resource lock error

Un appel a été effectué vers une procédure dans une ressource de code. La ressource de code a été trouvée, mais une erreur s'est produite lors de la tentative de verrouiller la ressource, probablement car elle est en cours d'utilisation.

Pour corriger cette erreur

- Vérifiez la ressource afin de déterminer l'endroit où l'erreur s'est produite.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Format de Presse-papiers non valide

Clipboard format is not valid

Le format de Presse-papiers spécifié est incompatible avec la méthode exécutée. Cette erreur peut se produire pour de nombreuses raisons.

- utilisation de la méthode **GetText** ou **SetText** du Presse-papiers avec un format de Presse-papiers autre que **vbCFText** ou **vbCFLink** ;
- utilisation de la méthode **GetData** ou **SetData** du Presse-papiers avec un format de Presse-papiers autre que **vbCFBitmap**, **vbCFDIB** ou **vbCFMetafile** ;
- utilisation de la méthode **DataObject GetData** ou **SetData** avec un format de Presse-papiers compris dans la plage réservée par Microsoft Windows pour les formats enregistrés (&HC000-&HFFFF), mais ce format de Presse-papiers n'a pas été enregistré par Microsoft Windows.

Pour corriger cette erreur

- Supprimez le format non valide et spécifiez un format valide.

Voir aussi

Concepts

Objet Clipboard pour les utilisateurs de Visual Basic 6.0
Clipboard: Adding Other Formats

L'index de la collection doit être compris entre 1 et la taille de la collection

Collection index must be in the range 1 to the size of the collection

La valeur d'index fournie se situe en dehors de la plage valide, comprise entre 1 et la taille de la collection.

Pour corriger cette erreur

- Remplacez la valeur fournie par une valeur valide.

Voir aussi

Concepts

Collection Controls pour les utilisateurs de Visual Basic 6.0

La connexion à la bibliothèque de types ou d'objets pour processus distant a été perdue

Connection to type library or object library for remote process has been lost

Vous avez perdu votre connexion à la bibliothèque d'objets ou de types du processus distant.

Pour corriger cette erreur

1. Redémarrez l'objet **Application**.
2. Dans la boîte de dialogue d'erreur qui vous a permis d'accéder à cette rubrique d'aide, cliquez sur **OK** pour afficher la boîte de dialogue **Références**. La référence perdue apparaît, précédée du mot "MANQUANT".
3. Supprimez la référence perdue.
4. Dans la boîte de dialogue **Références**, activez la case à cocher correspondant à l'objet **Application**.

Voir aussi

Concepts

Types d'erreurs

Impossible d'achever l'opération, car le répertoire cible se situe sous le répertoire source

Could not complete operation since target directory is under source directory

Une opération cyclique a échoué. Les opérations cycliques se répètent et, par conséquent, ne peuvent pas s'achever. Par exemple, l'objet A peut essayer d'hériter de l'objet B qui hérite, à son tour, de l'objet A.

Pour corriger cette erreur

- Durant un héritage, assurez-vous qu'il n'existe aucune référence circulaire.

Voir aussi

Concepts

Types d'erreurs

Concepts de base du débogage : points d'arrêt

Impossible d'obtenir des informations sur la mémoire en raison d'une erreur interne

Could not obtain memory information due to internal error

Un appel à l'une des propriétés d'informations sur la mémoire de l'objet **My.Computer.Info** a échoué.

Pour corriger cette erreur

- Ajoutez un bloc **Try...Catch** autour de l'appel à la propriété d'informations sur la mémoire de l'objet **My.Computer.Info**.

Voir aussi

Référence

My.Computer.Info, objet

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

Impossible d'obtenir le nom complet du système d'exploitation en raison d'une erreur interne

Message d'erreur
Impossible d'obtenir le nom complet du système d'exploitation en raison d'une erreur interne. Ceci peut être dû au fait que WMI est absent de l'ordinateur actuel.
Could not obtain full operation system name due to internal error. This might be caused by WMI not existing on the current machine.

Un appel à la propriété **My.Computer.Info.OSFullName** a échoué. Cet échec peut être dû au fait que Windows Management Instrumentation (WMI) n'est pas installé sur l'ordinateur actuel.

Pour corriger cette erreur

1. Ajoutez un bloc **Try...Catch** autour de l'appel à la propriété **My.Computer.Info.OSFullName**.
2. Pour plus d'informations sur WMI et comment l'installer, accédez à http://www.microsoft.com/france/support et recherchez « Windows Management Instrumentation Core ».

Voir aussi

Référence
My.Computer.Info.OSFullName, propriété
Try...Catch...Finally, instruction (Visual Basic)
Autres ressources
Gestion des exceptions et des erreurs en Visual Basic

DataBits doit être supérieur à 0

DataBits must be greater than 0

L'argument *DataBits* spécifié pour la méthode **My.Computer.Ports.OpenSerialPort** doit être supérieur à zéro.

Pour corriger cette erreur

- Remplacez la valeur de l'argument *DataBits* par une valeur positive.

Voir aussi

Référence

My.Computer.Ports.OpenSerialPort, méthode

Division par zéro (erreur d'exécution Visual Basic)

Division by zero (Visual Basic Run-Time Error)

Une expression utilisée en tant que diviseur possède une valeur égale à zéro.

Pour corriger cette erreur

1. Vérifiez l'orthographe des variables de l'expression. Un nom de variable mal orthographié peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur les variables de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

Division par zéro (erreur Visual Basic)

Division by zero (Visual Basic Error)

La valeur d'une expression utilisée en tant que diviseur est égale à zéro.

Pour corriger cette erreur

1. Vérifiez l'orthographe des variables de l'expression. Une variable mal orthographiée peut créer implicitement une variable numérique dont la valeur est zéro.
2. Vérifiez les opérations précédentes sur la variable de l'expression, particulièrement celles qui sont passées à la procédure en tant qu'arguments à partir d'autres procédures.

Voir aussi

Concepts

Types d'erreurs

Disque non prêt

Disk not ready

Il n'y a aucun disque dans le lecteur spécifié ou le volet du lecteur est ouvert.

Pour corriger cette erreur

1. Vérifiez le lecteur spécifié et insérez un disque si nécessaire.
2. Fermez le volet du lecteur.

Voir aussi

Concepts

Types d'erreurs

Disque plein (Visual Basic)

Disk full (Visual Basic)

L'espace disponible sur le disque est insuffisant pour terminer une opération **Print**, **Write** ou **FileClose** ou pour créer les fichiers requis.

Pour corriger cette erreur

- Augmentez l'espace disponible sur le disque en supprimant certains fichiers ou en les déplaçant vers un autre disque.

Voir aussi

Concepts

Types d'erreurs

La fonction 'Dir' doit d'abord être appelée avec un argument 'Pathname'

'Dir' function must first be called with a 'PathName' argument

Un appel initial à la fonction **Dir** n'inclut pas l'argument *PathName*. Le premier appel à **Dir** doit inclure *PathName*, mais les appels suivants à **Dir** ne doivent pas inclure de paramètres permettant d'extraire l'élément suivant.

Pour corriger cette erreur

- Fournissez un argument *PathName* dans l'appel de fonction.

Voir aussi

Référence

Dir, fonction

Périphérique non disponible

Device unavailable

Vous avez perdu votre connexion réseau ou le périphérique auquel vous tentez d'accéder est hors connexion ou n'existe pas.

Pour corriger cette erreur

1. Vérifiez l'alimentation du périphérique ainsi que les câbles le reliant à votre ordinateur. Si vous essayez d'accéder à une imprimante sur un réseau, vérifiez qu'une connexion logique existe entre votre ordinateur et l'imprimante, par exemple une connexion associant LPT1 à l'ID de l'imprimante réseau.
2. Reconnectez-vous au réseau et recommencez.

Voir aussi

Concepts

Types d'erreurs

Erreur d'E/S de périphérique

Device I/O error

Une erreur d'entrée ou de sortie est survenue lors de l'utilisation d'un périphérique (une imprimante ou un lecteur de disques, par exemple) par votre programme.

Pour corriger cette erreur

- Vérifiez que le périphérique fonctionne correctement, puis recommencez l'opération.

Voir aussi

Concepts

Types d'erreurs

Le lecteur '<nomlecteur>' est introuvable

Drive '<drivename>' not found

Le lecteur est introuvable. Soit votre connexion réseau a été perdue, soit le lecteur auquel vous tentez d'accéder est hors connexion ou n'existe pas.

Pour corriger cette erreur

1. Vérifiez l'alimentation du lecteur.
2. Vérifiez que les câbles reliant votre ordinateur au réseau sont correctement attachés.
3. Pour accéder à un lecteur sur un réseau, vérifiez qu'une connexion logique existe entre votre ordinateur et le lecteur.
4. Reconnectez-vous au réseau et recommencez.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Le codage ne peut pas avoir la valeur Nothing

Encoding cannot be set to Nothing

Une tentative de lecture ou d'écriture dans un fichier a échoué car le paramètre *encoding* a la valeur **Nothing**, mais requiert une valeur valide.

Encoding est utilisé pour déterminer le codage à utiliser lors de l'écriture dans un fichier. La valeur par défaut est UTF-8.

Pour corriger cette erreur

- Fournissez une valeur valide pour le paramètre *encoding*.

Voir aussi

Référence

My.Computer.FileSystem.ReadAllText, méthode
My.Computer.FileSystem.WriteAllText, méthode

Concepts

Codages de fichiers

Autres ressources

Lecture à partir de fichiers en Visual Basic
Écriture dans des fichiers en Visual Basic

Erreur de chargement de la DLL (Visual Basic)

Error in loading DLL (Visual Basic)

Une bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) est une bibliothèque spécifiée dans la clause **Lib** d'une instruction **Declare**. Cette erreur peut avoir plusieurs causes :

- Le fichier n'est pas un exécutable DLL.
- Le fichier n'est pas une DLL Microsoft Windows.
- La DLL fait référence à une autre DLL qui n'est pas présente.
- La DLL ou la DLL référencée n'est pas un répertoire spécifié par le chemin d'accès.

Pour corriger cette erreur

- Si le fichier est un fichier texte source et non un exécutable DLL, il doit être compilé et lié à un formulaire exécutable DLL.
- Si le fichier n'est pas une DLL Microsoft Windows, vous devez obtenir l'équivalent Microsoft Windows.
- Si la DLL fait référence à une autre DLL qui n'est pas présente, vous devez obtenir la DLL référencée et la rendre disponible.
- Si la DLL ou la DLL référencée n'est pas un répertoire spécifié par le chemin d'accès, déplacez la DLL vers un répertoire référencé.

Voir aussi

Référence

Declare, instruction

Le numéro d'erreur doit être compris entre 0 et 65 535

Error number must be within the range 0 and 65535

Un numéro d'erreur se situe en dehors de la plage autorisée comprise entre 0 et 65 535.

Pour corriger cette erreur

- Vérifiez que le numéro se situe dans la plage valide.

Voir aussi

Référence

Err, objet (Visual Basic)

Number, propriété (objet Err)

L'expression '<nom>' n'est pas une procédure, mais elle intervient en tant que cible d'un appel de procédure

Expression '<name>' is not a procedure, but occurs as the target of a procedure call

Vous avez effectué un appel de procédure vers une expression qui n'est pas une procédure.

Pour corriger cette erreur

1. Vérifiez que vous avez correctement orthographié le nom de la procédure.
2. Indiquez la procédure appelée par <*name*>.

Voir aussi

Concepts

Procédures dans Visual Basic

Cette fonctionnalité n'est pas encore implémentée

Feature not yet implemented

Vous avez tenté d'accéder à une fonctionnalité qui n'est pas actuellement implémentée dans Visual Basic 2005.

Pour corriger cette erreur

- Supprimez la référence.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Expression trop complexe

Expression too complex

Une expression à virgule flottante contient un trop grand nombre de sous-expressions imbriquées.

Pour corriger cette erreur

- Morcelez l'expression en plusieurs expressions séparées afin d'empêcher l'apparition de l'erreur.

Voir aussi

Concepts

Opérateurs et expressions en Visual Basic

Le champ '<nomchamp>' de type '<nomtype>' est 'ReadOnly'

Field '<fieldname>' of type '<typename>' is 'ReadOnly'

Vous avez tenté d'écrire dans un champ déclaré **ReadOnly**.

Pour corriger cette erreur

- Modifiez le champ afin qu'il ne soit plus déclaré **ReadOnly**.

Voir aussi

Référence

ReadOnly (Visual Basic)

Impossible de supprimer le fichier '<nomfichier>', car il est ouvert

File '<filename>' cannot be deleted because it is open

Vous avez tenté de supprimer un fichier ouvert.

Pour corriger cette erreur

- Fermez le fichier avant de le supprimer.

Voir aussi

Référence

FileClose, fonction
My.Computer.FileSystem.DeleteFile, méthode
My.Computer.FileSystem, objet

Le fichier '<nomfichier>' est protégé en écriture

File '<filename>' is write protected

Le fichier dans lequel vous essayez d'écrire est protégé en écriture et ne peut donc pas être modifié.

Pour corriger cette erreur

- Remplacez les autorisations du fichier par une autorisation de lecture/écriture ou écrivez dans un fichier différent.

Voir aussi

Tâches

Dépannage : lecture et écriture dans des fichiers texte

Le fichier '<nomfichier>' est introuvable

File '<filename>' not found

Un nom de fichier est incorrect ou le fichier n'existe pas.

Pour corriger cette erreur

- Vérifiez que le nom de fichier est correct et qu'il existe.

Voir aussi

Référence

My.Computer.FileSystem, objet

Fichier déjà existant

File already exists

Cette erreur se produit lors de l'attribution d'un nom d'un fichier existant à un nouveau fichier ou lors de l'utilisation de la commande **Save As** pour enregistrer un projet actuellement chargé sous un nom existant.

Pour corriger cette erreur

- Vérifiez le nom et remplacez-le par un nom qui ne provoque pas de conflit ou supprimez le fichier ou le projet existant s'ils ne sont plus nécessaires.

Voir aussi

Concepts

Types d'erreurs

Le fichier est déjà ouvert.

File already open

Un fichier doit parfois être fermé avant qu'une opération **FileOpen** ou qu'une autre opération ne se produise. Cette erreur peut se produire pour de nombreuses raisons.

- Une opération **FileOpen** de mode séquentiel Output a été exécutée pour un fichier déjà ouvert.
- Une instruction fait référence à un fichier ouvert.

Pour corriger cette erreur

- Fermez le fichier avant d'exécuter l'instruction.

Voir aussi

Référence

FileOpen, fonction

Le format de fichier n'est pas valide

File format not valid

Vous avez tenté de charger un fichier dans un composant, mais le format des données de ce fichier était incompatible avec le composant, ou vous avez tenté d'enregistrer des données de composant dans un fichier, mais le format des données était incompatible avec le format du fichier.

Pour corriger cette erreur

1. Consultez la documentation du composant afin de déterminer le format correct des données du fichier disque.
2. Consultez la documentation du composant afin de déterminer s'il assure la prise en charge d'une conversion d'un format en un autre.

Voir aussi

Concepts

Types d'erreurs

L'E/S de fichier d'une structure avec un champ '<nomchamp>' de type '<nomtype>' n'est pas valide

File I/O of a structure with field '<fieldname>' of type '<typename>' is not valid

La structure dans laquelle vous tentez d'écrire dans un fichier contient un type non pris en charge.

Pour corriger cette erreur

- Vérifiez la structure afin de déterminer l'endroit où s'est produit le type non pris en charge, puis remplacez-le par un type valide.

Voir aussi

Référence

My.Computer.FileSystem.WriteAllBytes, méthode

Concepts

Types d'erreurs

L'E/S de fichier de type '<nomtype>' n'est pas valide

File I/O with type '<typename>' is not valid

Vous essayez d'écrire un type non pris en charge dans un fichier.

Pour corriger cette erreur

- Vérifiez la structure afin de déterminer l'endroit où s'est produit le type non pris en charge, puis remplacez-le par un type valide.

Voir aussi

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.WriteAllBytes, méthode

Impossible d'interroger les informations du fichier si ce dernier n'existe pas

File information cannot be queried if the file does not exist

Une tentative a été effectuée pour interroger un objet FileInfo pour obtenir des informations sur un fichier qui n'existe pas. Cela est généralement dû à une erreur lors de la saisie du nom du fichier.

Pour corriger cette erreur

- Vérifiez que vous entrez le nom de fichier correct.

Voir aussi

Tâches

Comment : obtenir des informations à propos d'un fichier dans Visual Basic

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem, objet

Impossible d'interroger les informations du fichier lorsque ce dernier est ouvert en écriture

File information cannot be queried while open for writing

Une tentative a été effectuée pour accéder aux informations sur un fichier qui est actuellement ouvert en écriture.

Pour corriger cette erreur

- Fermez le fichier et recommencez l'opération.

Voir aussi

Tâches

Comment : obtenir des informations à propos d'un fichier dans Visual Basic

Référence

FileInfo

FileIOPermission

Le fichier est trop volumineux pour être lu dans un tableau d'octets

File is too large to read into a byte array

La taille du fichier que vous essayez de lire dans un tableau d'octets est supérieure à 4 Go. La méthode **My.Computer.FileSystem.ReadAllBytes** ne peut pas lire un fichier qui dépasse cette taille.

Pour corriger cette erreur

- Utilisez StreamReader pour lire le fichier. Pour plus d'informations, consultez Concepts de base du système de fichiers et des E/S de fichier du .NET Framework.

Voir aussi

Tâches

Comment : lire le texte des fichiers avec un StreamReader (Visual Basic)

Référence

My.Computer.FileSystem.ReadAllBytes, méthode

StreamReader

Autres ressources

Accès au fichier avec Visual Basic

Nom du fichier ou de la classe introuvable lors de l'opération Automation (Visual Basic)

File name or class name not found during Automation operation (Visual Basic)

Le nom spécifié pour le nom de fichier ou la classe dans un appel à la fonction **GetObject** est introuvable.

Pour corriger cette erreur

- Vérifiez les noms, puis recommencez. Vérifiez que le nom utilisé pour le paramètre *class* correspond à celui enregistré dans le système.

Voir aussi

Concepts

Types d'erreurs

Fichier introuvable (erreur d'exécution Visual Basic)

File not found (Visual Basic Run-Time Error)

Le fichier n'a pas été trouvé à l'endroit spécifié. Les causes de cette erreur peuvent être les suivantes :

- Une instruction fait référence à un fichier qui n'existe pas.
- Une tentative a été effectuée d'appeler une procédure dans une bibliothèque de liens dynamiques (DLL, Dynamic-Link Library), mais la bibliothèque spécifiée dans la clause **Lib** de l'instruction **Declare** est introuvable.
- Vous avez tenté d'ouvrir un projet ou de charger un fichier texte qui n'existe pas.

Pour corriger cette erreur

- Vérifiez l'orthographe du nom de fichier et la spécification du chemin d'accès.

Voir aussi

Référence

Declare, instruction

Le fichier spécifié par FileName n'utilise pas le codage spécifié par FileEncoding

File specified by FileName does not use the encoding specified by FileEncoding

Une tentative a été effectuée pour lire un fichier, mais le codage spécifié n'est pas utilisé par le fichier.

Pour corriger cette erreur

- Vérifiez que le codage de fichier correct est fourni.

Voir aussi

Tâches

Comment : lire des fichiers texte dans Visual Basic

Référence

My.Computer.FileSystem.ReadAllText, méthode

My.Computer.FileSystem, objet

Concepts

Codages de fichiers

Le fichier spécifié dans le NomFichier n'est pas un fichier XML valide

File specified in FileName is not a valid XML file

Le nom de fichier que vous avez fourni n'est pas un fichier XML valide. Pour spécifier la structure et le contenu autorisés d'un document XML, vous pouvez utiliser une définition de type de document (DTD), un schéma XDR (XML-Data Reduced) de Microsoft, ou un schéma de langage XSD (XML Schema definition). Les schémas XSD représentent la méthode préférée pour spécifier des grammaires XML dans le .NET Framework.

Remarque

Dans les versions précédentes de Visual Studio 2005, le **Concepteur XML** représente le concepteur des groupes de données typés et du schéma XML. Le **Concepteur XML** peut encore être utilisé pour créer et modifier les fichiers de schéma XML. Tou tefois, dans Visual Studio 2005, le **Concepteur de DataSet** représente le concepteur de la création et de la modification des groupes de données typés. Pour plus d'informations, consultez Concepteur de DataSet.

Pour corriger cette erreur

- Vérifiez que le nom du fichier fourni est correct.
- Vérifiez que le fichier spécifié contient du code XML valide en chargeant le fichier XML que vous souhaitez vérifier dans le **Concepteur XML**. Dans le menu **XML**, sélectionnez **Valider les données XML**. Affichez toutes les erreurs dans la **Liste des tâches**.
- Si le fichier XML est associé à un schéma, vérifiez que ses éléments apparaissent dans la structure définie et que le contenu de chaque élément est conforme aux types de données déclarés spécifiés dans le schéma.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

System.Xml

Le mode d'ouverture du fichier n'a pas été défini à une valeur valide

File's open mode wasn't set to a valid value

La valeur fournie pour le mode d'ouverture du fichier n'est pas valide. Le tableau suivant présente des valeurs valides pour l'énumération OpenMode.

Valeur	Mode
1	**OpenMode.Input**
2	**OpenMode.Output**
4	**OpenMode.Random**
8	**OpenMode.Append**
32	**OpenMode.Binary**

Pour corriger cette erreur

- Vérifiez la valeur fournie pour le mode d'ouverture du fichier.

Voir aussi

Référence

OpenMode, énumération

My.Computer.FileSystem, objet

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

Boucle For non initialisée (Visual Basic)

For loop not initialized (Visual Basic)

Vous êtes arrivé au milieu d'une boucle **For...Next**.

Pour corriger cette erreur

- Supprimez le saut dans la boucle.

Voir aussi

Tâches

Comment : étiqueter des instructions

Référence

For Each...Next, instruction (Visual Basic)

Le format n'est pas valide dans le fichier de ressources

Format not valid in resource file

Un fichier de ressources de votre projet ne respecte pas le format standard d'un fichier de ressources Windows ou une incompatibilité existe entre le fichier de ressources et le projet Visual Basic.

Pour corriger cette erreur

1. Recréez la source de la ressource et les fichiers de ressources et respectez la syntaxe documentée dans le Kit de développement logiciel (SDK) Windows.
2. Supprimez le fichier de ressources non valide du projet.

Voir aussi

Concepts

Types d'erreurs

Get non pris en charge (propriété en écriture seule)

Get not supported (write-only property)

Une propriété ne peut pas être lue, car elle est en écriture seule.

Pour corriger cette erreur

1. Vérifiez la propriété et déterminez les conditions sous lesquelles elle peut être définie.
2. Supprimez la référence à la propriété.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Get non pris en charge au moment de l'exécution

Get not supported at run time

Vous avez tenté de lire une propriété au moment de l'exécution qui est seulement accessible au moment du design.

Pour corriger cette erreur

1. Vérifiez la propriété et déterminez les conditions sous lesquelles elle peut être définie.
2. Supprimez la référence à la propriété.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Toute référence implicite à un objet en construction n'est pas valide lors de l'appel d'un autre constructeur

Implicit reference to object under construction is not valid when calling another constructor

Une référence à un membre objet a été faite avant que le constructeur de l'objet n'ait terminé la construction de l'objet.

ID d'erreur : BC31096

Pour corriger cette erreur

- N'utilisez pas **MyBase**, **MyClass** ou **Me** lorsque vous appelez un constructeur à partir d'un autre constructeur.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

Concepts

Durée de vie d'un objet : création et destruction des objets

Autorisations de sécurité insuffisantes pour définir l'heure système

Insufficient security permissions to set the system time

Les autorisations de sécurité sont insuffisantes pour définir l'heure système.

Pour corriger cette erreur

- Modifiez les autorisations de sécurité.

Voir aussi

Référence

System.Security.Permissions Namespace

Concepts

Niveaux de stratégie de sécurité

Autres ressources

Configuration de la stratégie de sécurité à l'aide de l'outil .NET Framework Configuration Tool (Mscorcfg.msc)

Autorisations de sécurité insuffisantes pour définir la date système

Insufficient security permissions to set the system date

Les autorisations de sécurité sont insuffisantes pour définir la date système.

Pour corriger cette erreur

- Modifiez les autorisations de sécurité afin d'autoriser la définition de la date système.

Voir aussi

Référence

System.Security.Permissions

Concepts

Niveaux de stratégie de sécurité

Autres ressources

Configuration de la stratégie de sécurité à l'aide de l'outil .NET Framework Configuration Tool (Mscorcfg.msc)

L'entrée dépasse la fin du fichier

Input past end of file

Une instruction **Input** lit à partir d'un fichier vide ou d'un fichier dans lequel toutes les données sont utilisées ou vous avez utilisé la fonction **EOF** avec un fichier ouvert pour un accès binaire.

Pour corriger cette erreur

1. Utilisez la fonction **EOF** immédiatement avant l'instruction **Input** pour détecter la fin du fichier.
2. Si le fichier est ouvert pour un accès binaire, utilisez **Seek** et **Loc**.

Voir aussi

Référence

Input, fonction
EOF, fonction
Seek, fonction
Loc, fonction

Erreur interne (Visual Basic)

Internal error (Visual Basic)

Une erreur interne s'est produite dans Visual Basic.

Pour corriger cette erreur

- Vérifiez que cette erreur n'a pas été générée par l'instruction **Error** ou la méthode **Raise**. Si tel n'est pas le cas, contactez les services de Support Technique de Microsoft pour signaler les circonstances selon lesquelles ce message est apparu.

Voir aussi

Référence

On Error, instruction (Visual Basic)
Raise, méthode (objet Err)

Une erreur interne s'est produite dans <emplacement>

Internal error happened at <location>

Une erreur interne s'est produite. La ligne à laquelle elle s'est produite est indiquée dans le message d'erreur.

Pour corriger cette erreur

- Assurez-vous que cette erreur n'a pas été générée par l'instruction **Error** ou la méthode **Raise** ; si tel n'est pas le cas, contactez les services de Support Technique de Microsoft pour signaler les conditions dans lesquelles le message est apparu.

Voir aussi

Autres ressources

Présentation du débogueur

Erreur interne dans le runtime Microsoft Visual Basic

Internal error in the Microsoft Visual Basic runtime

Une erreur interne s'est produite dans le runtime Microsoft Visual Basic.

Pour corriger cette erreur

- Prenez note des circonstances dans lesquelles l'erreur est survenue et appelez les services de Support Technique de Microsoft.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Chaîne de masque non valide

Invalid pattern string

La chaîne de masque spécifiée dans l'opération **Like** d'une recherche n'est pas valide.

Pour corriger cette erreur

1. Examinez les caractères valides des expressions de liste.
2. Dans la plage de masques, vérifiez que le caractère de début de plage est inférieur au caractère de fin de plage, comme dans `[a-z]`.
3. Dans la plage de masques, vérifiez que plusieurs traits d'union ne se situent pas les uns à côté des autres, comme dans `[a--z]`.
4. Terminez les plages de masques par un crochet fermant.

Voir aussi

Référence

Like, opérateur

La clé ne peut pas être supprimée car elle a des sous-clés

Key cannot be deleted because it has subkeys

La clé ne peut pas être supprimée car elle a des sous-clés.

Pour corriger cette erreur

- Déterminez les sous-clés de la clé spécifiée et supprimez-les avant d'effectuer cette opération.

Voir aussi

Tâches

Comment : supprimer une clé de Registre dans Visual Basic

Dépannage : manipulation du Registre

Référence

DeleteSubKey

Concepts

Lecture et écriture dans le Registre à l'aide de My

Assignation de liaison tardive à un champ de type valeur '<nomtype>' non valide lorsque '<nom>' représente le résultat d'une expression de liaison tardive

Late-bound assignment to a field of value type '<typename>' is not valid when '<name>' is the result of a late-bound expression

Vous avez tenté une assignation de liaison tardive qui n'est pas valide.

Pour corriger cette erreur

- Effectuez une assignation à liaison anticipée.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

La longueur de l'argument '<NomArgument>' doit être supérieure à zéro

Length of argument '<argumentname>' must be greater than zero

Un argument possède une valeur égale ou inférieure à zéro.

Pour corriger cette erreur

- Fournissez un argument supérieur à zéro.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

La ligne <numéro> ne peut pas être analysée à l'aide du Delimiters actuel

Line <number> cannot be parsed using the current Delimiters

La ligne spécifiée ne peut pas être analysée parce qu'elle utilise des délimiteurs autres que ceux spécifiés.

Pour corriger cette erreur

- Modifiez **Delimiters** de sorte que la ligne puisse être analysée correctement, ou insérez le code de gestion des exceptions pour gérer la ligne.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem.OpenTextFieldParser, méthode
TextFieldParser, objet
TextFieldParser.Delimiters, propriété
TextFieldParser.SetDelimiters, méthode

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

La ligne <numéro> ne peut pas être analysée à l'aide du FieldWidths actuel

Line <number> cannot be parsed using the current FieldWidths

La ligne spécifiée ne peut pas être analysée car ses champs ont des largeurs différentes de celles spécifiées.

Pour corriger cette erreur

- Modifiez **FieldWidths** de sorte que la ligne puisse être analysée correctement, ou insérez le code de gestion des exceptions pour gérer la ligne.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem.OpenTextFieldParser, méthode

TextFieldParser, objet

TextFieldParser.FieldWidths, propriété

TextFieldParser.SetFieldWidths, méthode

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

La ligne <numéro> ne peut pas être lue, car elle dépasse la taille de ligne maximale

Line <number> cannot be read because it exceeds the maximum line size.

La ligne ne peut pas être lue, car elle dépasse la taille de ligne maximale. Généralement, cela signifie qu'il y a un guillemet d'ouverture dans le fichier, mais pas de guillemet de fermeture.

Pour corriger cette erreur

- Insérez un guillemet de fermeture dans le fichier.

Voir aussi

Référence

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

L'ID de paramètres régionaux '<nom>' n'est pas pris en charge sur ce système

Locale ID '<name>' is not supported on this system

Vous essayez d'utiliser **VbStrConv**, mais votre système ne prend pas en charge les paramètres régionaux spécifiés.

Pour corriger cette erreur

- Procédez à une mise à niveau de l'ordinateur afin d'assurer la prise en charge des paramètres régionaux ou exécutez l'application sur un ordinateur qui prend en charge les paramètres régionaux spécifiés.

Voir aussi

Référence

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

La variable contrôle de boucle de type '<nomtype>' n'implémente pas l'interface 'System.IComparable'

Loop control variable of type '<typename>' does not implement the 'System.IComparable' interface

Une variable possède un type qui n'implémente pas l'interface **System.IComparable**. **System.IComparable** doit être implémentée par un type valeur ou une classe.

Pour corriger cette erreur

- Vérifiez que la variable est un type valeur.

Voir aussi

Référence

IComparable Interface

Les classes managées dérivées d'une classe COM ne peuvent pas être appelées à liaison tardive

Managed classes derived from a COM class cannot be called late-bound.

Vous avez tenté d'effectuer un appel à liaison tardive vers une classe managée dérivée d'une classe COM ; ces appels ne sont pas pris en charge.

Pour résoudre ce problème

- Effectuez un appel à liaison anticipée.

Voir aussi

Concepts

Types d'erreurs

La méthode '<nomméthode>' ne peut pas être appelée avec <nombre> arguments

Method '<methodname>' cannot be called with <number> arguments

Un appel de méthode contient un nombre incorrect d'arguments.

Pour corriger cette erreur

1. Comptez les arguments fournis.
2. Faites correspondre le nombre au nombre de paramètres de la méthode que vous souhaitez appeler.
3. Ajustez le nombre d'arguments si les deux nombres ne correspondent pas.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

La méthode '<nomméthode>' n'a aucun paramètre nommé '<nomparamètre>'

Method '<methodname>' has no parameter named '<parametername>'

Un argument nommé fourni ne correspond à aucun argument de la méthode cible.

Pour corriger cette erreur

- Vérifiez que l'argument nommé a été correctement fourni.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

Méthode ou données membres introuvables

Method or data member not found

La méthode ou les données membres auxquelles vous avez essayé d'accéder sont introuvables.

Pour corriger cette erreur

- Vérifiez que vous avez correctement orthographié le nom de la méthode ou des données membres.

Voir aussi

Concepts

Types d'erreurs

My.Application.Log ne peut pas déterminer la quantité d'espace disque libre

My.Application.Log cannot determine the amount of free disk space

La classe FileLogTraceListener ne peut pas déterminer la quantité d'espace disque libre.

Pour corriger cette erreur

- Prenez note des circonstances dans lesquelles l'erreur est survenue et appelez les services de Support Technique de Microsoft.

Voir aussi

Référence

FileLogTraceListener

Autres ressources

Support technique

Support technique et accessibilité

L'argument nommé '<nomargument>' a été spécifié plusieurs fois

Named argument '<argumentname>' specified multiple times

Vous avez spécifié plusieurs fois un argument nommé.

Pour corriger cette erreur

- Conservez une seule spécification et supprimez toutes les autres.

Voir aussi

Concepts

Passage des arguments par position et par nom

Argument nommé introuvable (Visual Basic)

Named argument not found (Visual Basic)

Un argument nommé ne peut pas être utilisé dans un appel de procédure, sauf s'il apparaît dans la définition de procédure. Vous avez spécifié un argument nommé, mais la procédure n'a pas été définie pour accepter un argument de ce nom.

Pour corriger cette erreur

- Vérifiez l'orthographe du nom de l'argument.

Voir aussi

Concepts

Types d'erreurs

Les arguments nommés ne peuvent pas correspondre aux paramètres ParamArray

Named arguments cannot match ParamArray parameters

Le tableau de paramètres doit être passé par valeur.

Pour corriger cette erreur

- Passez les arguments **ByVal**, au lieu de leur attribuer un nom.

Voir aussi

Concepts

Tableaux de paramètres
Passage d'arguments par valeur et par référence
Passage des arguments par position et par nom

Index de tableau de propriétés requis

Need property array index

Cette valeur de propriété est constituée d'un tableau au lieu d'une seule valeur. Vous n'avez pas spécifié l'index du tableau de propriétés auquel vous avez tenté d'accéder.

Pour corriger cette erreur

- Consultez la documentation du composant pour trouver la plage des index appropriés pour le tableau. Spécifiez un index approprié dans votre instruction d'accès à la propriété.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Aucun '<nomméthode>' surchargé accessible ne peut être appelé avec ces arguments sans conversion restrictive

No accessible overloaded '<methodname>' can be called with these arguments without a narrowing conversion

Une méthode surchargée a été appelée, mais aucune méthode n'a été mise en correspondance avec la liste d'arguments fournis sans conversion restrictive.

Pour corriger cette erreur

1. Spécifiez **Option Strict Off**.
2. Modifiez les arguments pour les faire correspondre aux signatures de la méthode surchargée.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Conversions étendues et restrictives

Aucun '<nomméthode>' surchargé accessible ne peut être appelé avec ces arguments sans conversion restrictive : <liste>

No accessible overloaded '<methodname>' can be called with these arguments without a narrowing conversion: <list>

Une méthode surchargée a été appelée, mais aucune méthode n'a été mise en correspondance avec la liste d'arguments fournis sans conversion restrictive.

Pour corriger cette erreur

1. Spécifiez **Option Strict Off**.
2. Modifiez les arguments pour les faire correspondre à une signature de la méthode surchargée.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Conversions étendues et restrictives

Aucun '<nomméthode>' surchargé accessible ne peut être appelé avec ces arguments sans conversion étendue : <liste>

No accessible overloaded '<methodname>' can be called with these arguments without a widening conversion: <list>

Une méthode surchargée a été appelée, mais aucune méthode n'a pu être mise en correspondance avec la liste d'arguments fournis sans conversion étendue.

Pour corriger cette erreur

- Spécifiez **Option Strict Off**.
- Modifiez les arguments pour les faire correspondre aux signatures de la méthode surchargée.

Voir aussi

Référence

Option Strict, instruction

Concepts

Conversions étendues et restrictives

Aucun '<nomméthode>' surchargé accessible ne peut être appelé avec ces arguments : <liste>

No accessible overloaded '<methodname>' can be called with these arguments: <list>

Une méthode surchargée est appelée, mais aucune méthode ne peut correspondre à la liste d'arguments fournie.

Pour corriger cette erreur

- Faites correspondre la liste d'arguments à la surcharge que vous essayez d'appeler.
- Ajustez la liste d'arguments pour faire correspondre la signature à la surcharge.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

Aucun membre par défaut trouvé pour le type '<nomtype>'

No default member found for type '<typename>'

Un appel **Get** ou **Set** à un index à liaison tardive a été tenté.

Pour corriger cette erreur

- Vérifiez que l'objet possède une propriété par défaut avec **Get** ou **Set**, en fonction de celui que vous essayez de créer.

Voir aussi

Référence

Get, instruction

Set, instruction (Visual Basic)

Aucun fichier correspondant à '<nomfichier>' n'a été trouvé

No files found matching '<filename>'

Le nom de fichier est incorrect ou le fichier n'existe pas.

Pour corriger cette erreur

- Assurez-vous que le nom de fichier est correct.

Voir aussi

Référence

My.Computer.FileSystem, objet

Aucune méthode '<nomméthode>' n'accepte un argument de type '<nomtype>' pour le paramètre '<nomparamètre>'

No method '<methodname>' can accept an argument of type '<typename>' for parameter '<parametername>'

Un appel de méthode contient un argument dont le type n'est pas valide.

Pour corriger cette erreur

1. Vérifiez les types des arguments.
2. Faites correspondre les types d'arguments aux types de paramètres de la méthode pour vous assurer qu'ils concordent. Modifiez les types le cas échéant.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence

Absence de souris

No mouse is present

L'une des propriétés de l'objet **My.Computer.Mouse** a été appelée, mais aucune souris ou port souris n'est installé sur l'ordinateur.

Pour corriger cette erreur

- Ajoutez un bloc **Try...Catch** autour de l'appel à la propriété de l'objet **My.Computer.Mouse**.

  - ou -

- Installez une souris sur l'ordinateur.

Voir aussi

Référence

My.Computer.Mouse, objet

Try...Catch...Finally, instruction (Visual Basic)

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

Absence de roulette de souris

No mouse wheel is present

La propriété **My.Computer.Mouse.WheelScrollLines** a été appelée, mais la souris n'a aucune roulette de défilement.

Pour corriger cette erreur

- Vérifiez la propriété **My.Computer.Mouse.WheelExists** pour savoir si la souris a une roulette de défilement avant d'appeler la propriété **My.Computer.Mouse.WheelScrollLines**.

  - ou -

- Installez sur l'ordinateur une souris avec une roulette de défilement.

Voir aussi

Référence

My.Computer.Mouse.WheelScrollLines, propriété

My.Computer.Mouse.WheelExists, propriété

Autres ressources

Gestion des exceptions et des erreurs en Visual Basic

NumberOfChars doit être supérieur à zéro

NumberOfChars must be greater than zero

Lorsque vous utilisez la méthode **PeekChars** de l'objet **TextFieldParser**, vous devez fournir une valeur *NumberOfChars* qui est supérieure à `0`.

Pour corriger cette erreur

- Remplacez *NumberOfChars* par une valeur supérieure à `0`.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem.OpenTextFieldParser, méthode

TextFieldParser.PeekChars, méthode

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

L'objet ne prend pas en charge les paramètres régionaux actuels (Visual Basic)

Object doesn't support current locale setting (Visual Basic)

Vous avez tenté d'accéder à un objet qui ne prend pas en charge les paramètres régionaux du projet actuel.

Pour corriger cette erreur

- Vérifiez les paramètres régionaux pris en charge par l'objet.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Cet objet ne prend pas en charge les arguments nommés

Object doesn't support named arguments

Vous avez tenté d'accéder à un objet dont les méthodes ne prennent pas en charge les arguments nommés.

Pour corriger cette erreur

- Spécifiez les arguments par position lors de l'exécution de méthodes sur cet objet. Consultez la documentation de l'objet pour plus d'informations sur les positions et les types d'arguments.

Voir aussi

Concepts

Types d'erreurs

L'objet ne prend pas en charge cette action (Visual Basic)

Object doesn't support this action (Visual Basic)

Vous avez référencé une méthode ou une propriété qui n'est pas prise en charge par cet objet.

Pour corriger cette erreur

1. Consultez la documentation de l'objet pour plus d'informations sur l'objet.
2. Assurez-vous que les noms des propriétés et des méthodes sont correctement orthographiés.

Voir aussi

Concepts

Types d'erreurs

L'objet ne prend pas en charge cette propriété ou méthode (Visual Basic)

Object doesn't support this property or method (Visual Basic)

Vous avez spécifié une méthode ou une propriété qui n'existe pas pour cet objet Automation.

Pour corriger cette erreur

- Consultez la documentation de l'objet pour plus d'informations sur celui-ci et vérifiez l'orthographe des propriétés et des méthodes.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

L'objet ou la classe ne prend pas en charge le jeu d'événements

Object or class does not support the set of events

Vous avez essayé d'utiliser une variable **WithEvents** avec un composant qui ne peut pas fonctionner comme une source d'événement pour le jeu d'événements spécifié. Par exemple, vous souhaitiez recevoir les événements d'un objet, puis créer un autre objet qui **Implements** le premier objet. Même si vous pensez pouvoir recevoir les événements de l'objet implémenté, ce n'est pas toujours le cas. **Implements** implémente uniquement une interface pour les méthodes et les propriétés. **WithEvents** n'est pas pris en charge pour les **UserControls** privés, car les informations sur le type requises pour déclencher **ObjectEvent** ne sont pas disponibles au moment de l'exécution.

Pour corriger cette erreur

- Vous ne pouvez pas recevoir d'événements pour un composant qui ne fournit pas d'événements.

Voir aussi

Référence

WithEvents

Implements, instruction

Objet requis (Visual Basic)

Object required (Visual Basic)

Les références à des propriétés et à des méthodes nécessitent souvent un qualificateur d'objet explicite. C'est le cas ici.

Pour corriger cette erreur

1. Vérifiez que les références à une propriété ou à une méthode d'objet ont un qualificateur d'objet valide. Spécifiez un qualificateur d'objet si vous n'en avez pas fourni.
2. Vérifiez l'orthographe du qualificateur d'objet et assurez-vous que l'objet est visible dans la partie du programme où vous y faites référence.
3. Si un chemin d'accès est fourni vers la commande **Fichier Ouvrir** d'une application hôte, vérifiez que les arguments qui s'y trouvent sont corrects.
4. Consultez la documentation de l'objet et assurez-vous que l'action est valide.

Voir aussi

Concepts

Types d'erreurs

Variable objet ou variable bloc With non définie

Object variable or With block variable not set

Une variable non valide est référencée. Pour créer une variable objet, déclarez-la, puis assignez-lui une référence valide à l'aide de l'instruction **Set**. De même, un bloc **With...End With** doit être initialisé en exécutant le point d'entrée de l'instruction **With**.

Pour corriger cette erreur

1. Vérifiez que la variable objet référence bien un objet valide et spécifiez (ou spécifiez de nouveau) une référence pour l'objet.
2. Vérifiez que vous n'avez pas utilisé de variable objet avec pour valeur **Nothing**.
3. Vérifiez que la bibliothèque d'objets dans laquelle l'objet a été décrit a été sélectionnée dans la boîte de dialogue **Add References**.
4. Vérifiez que votre bloc **With** s'est initialisé en exécutant le point d'entrée de l'instruction **With**.

Voir aussi

Référence

With...End With, instruction (Visual Basic)

Concepts

Déclaration des variables objets

Un ou plusieurs dossiers dans le chemin d'accès cible n'existent pas

One or more folders in the target path do not exist

Un ou plusieurs dossiers des répertoires fournis dans le chemin d'accès cible n'existent pas. Cela peut être dû à un chemin d'accès analysé incorrectement, surtout si vous fournissez un chemin d'accès relatif.

Pour corriger cette erreur

- Vérifiez que vous fournissez le chemin d'accès cible correct.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Seuls les premiers huit caractères sont pris en compte dans un nom de journal personnalisé

Only the first eight characters of a custom log name are significant

Lors de la vérification de l'unicité des noms des journaux des événements, seuls les huit premiers caractères sont pris en compte. Un conflit peut provenir des journaux des événements qui partagent leurs huit premiers caractères.

Pour corriger cette erreur

- Attribuez au journal des événements un nom dont les huit premiers caractères sont uniques.

Voir aussi

Tâches

Comment : créer et supprimer des journaux des événements personnalisés

Autres ressources

Gestion des journaux des événements

L'opérateur n'est pas valide pour le type '<nomtype>'

Operator is not valid for type '<typename>'

Un opérateur est inapproprié pour les types.

Pour corriger cette erreur

- Vérifiez que l'opérateur est approprié pour les types manipulés.

Voir aussi

Autres ressources

Opérateurs (Visual Basic)

L'opérateur n'est pas valide pour '<nom1>' et '<nom2>'

Operator is not valid for '<name1>' and '<name2>'

Un opérateur est inapproprié pour les types manipulés.

Pour corriger cette erreur

- Vérifiez que l'opérateur est approprié pour les types manipulés.

Voir aussi

Autres ressources

Opérateurs (Visual Basic)

Dépassement de capacité (erreur d'exécution Visual Basic)

Overflow (Visual Basic Run-Time Error)

Un dépassement de capacité se produit lorsque vous tentez d'effectuer une assignation qui dépasse les limites de la cible de l'assignation.

Pour corriger cette erreur

1. Vérifiez que les résultats des assignations, calculs et conversions de types de données ne sont pas trop grands pour être représentés dans la plage de variables autorisées pour ce type de valeur et assignez la valeur à une variable d'un type pouvant contenir une plage plus étendue de valeurs, si nécessaire.
2. Vérifiez que les assignations aux propriétés correspondent à la plage de la propriété dans laquelle elles sont effectuées.
3. Vérifiez que les nombres utilisés dans les calculs et convertis en entiers n'ont pas de résultats supérieurs aux entiers.

Voir aussi

Concepts

Types d'erreurs

Espace de chaîne insuffisant (Visual Basic)

Out of string space (Visual Basic)

Visual Basic 2005 vous permet d'utiliser de très grandes chaînes. Toutefois, les exigences d'autres programmes et la manière dont vous manipulez vos chaînes peuvent provoquer cette erreur.

Pour corriger cette erreur

1. Vérifiez qu'une expression nécessitant la création d'une chaîne temporaire lors de l'évaluation ne déclenche pas l'erreur.
2. Supprimez de la mémoire toutes les applications dont vous n'avez pas besoin afin de créer de l'espace supplémentaire.

Voir aussi

Référence

Liste des manipulations de chaînes

Concepts

Types d'erreurs

Espace de pile insuffisant (Visual Basic)

Out of stack space (Visual Basic)

La pile est une zone de mémoire dont la taille augmente ou diminue de manière dynamique en fonction des besoins du programme en cours d'exécution. Ses limites ont été dépassées.

Pour corriger cette erreur

1. Vérifiez que les procédures ne sont pas imbriquées trop profondément.
2. Assurez-vous que les procédures récursives se terminent correctement.
3. Si les variables locales requièrent davantage d'espace que la quantité disponible, essayez de déclarer certaines variables au niveau du module. Vous pouvez également déclarer toutes les variables de la procédure comme statiques en faisant précéder le mot clé **Property**, **Sub** ou **Function** de **Static**. Vous pouvez aussi utiliser l'instruction **Static** pour déclarer des variables statiques individuelles à l'intérieur de procédures.
4. Étant donné que les chaînes de longueur fixe utilisent davantage d'espace de pile que les chaînes de longueur variable, redéfinissez certaines chaînes de longueur fixe en chaînes de longueur variable. Vous pouvez également définir la chaîne au niveau du module où elle ne requiert aucun espace de pile.
5. Vérifiez le nombre d'appels de fonctions **DoEvents** imbriquées à l'aide de la boîte de dialogue **Calls** pour afficher les procédures actives sur la pile.
6. Vérifiez que vous n'avez provoqué aucune cascade d'événements en déclenchant un événement qui appelle une procédure événementielle qui se trouve déjà sur la pile. Une cascade d'événements est similaire à un appel de procédure récursive inachevé, mais elle est moins évidente, compte tenu que l'appel est effectué par Visual Basic et qu'il ne s'agit donc pas d'un appel explicite dans le code. Utilisez la boîte de dialogue **Calls** pour afficher les procédures actives sur la pile.

Voir aussi

Concepts

Mémoire, fenêtres

Mémoire insuffisante (Erreur d'exécution Visual Basic)

Out of memory (Visual Basic Run-Time Error)

Une mémoire supérieure à celle disponible est requise.

Pour corriger cette erreur

1. Fermez les applications, documents ou fichiers sources dont vous n'avez pas besoin.
2. Divisez les modules ou procédures de grande taille en éléments de plus petite taille.
3. Éliminez les programmes résidents.
4. Éliminez les pilotes de périphériques non nécessaires.
5. Réduisez le nombre de variables publiques.

Voir aussi

Concepts

Types d'erreurs

Nombre non valide

Ordinal is not valid

Votre appel à une bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) spécifie d'utiliser un nombre au lieu d'un nom de procédure, à l'aide de la syntaxe **#num**. Les causes de cette erreur peuvent être les suivantes :

- Une tentative de convertir l'expression **#num** en un nombre a échoué.
- L'expression **#num** spécifiée n'indique pas de fonction dans la DLL.
- Une bibliothèque de types possède une déclaration non valide entraînant une utilisation interne d'un nombre ordinal non valide.

Pour corriger cette erreur

1. Vérifiez que l'expression représente un nombre valide ou appelez la procédure par le nom.
2. Vérifiez que **#num** identifie une fonction valide dans la DLL.
3. Isolez l'appel de procédure générant le problème en commentant le code. Écrivez une instruction **Declare** pour la procédure et signalez le problème au fournisseur de la bibliothèque de types.

Voir aussi

Référence

Declare, instruction

Le chemin d'accès '<nomcheminaccès>' est introuvable

Path '<pathname>' not found

Le système d'exploitation n'est pas en mesure de trouver le chemin d'accès spécifié lors d'une opération d'accès au fichier ou au disque. Le chemin d'accès à un fichier comprend la spécification de lecteur et les répertoires et sous-répertoires qui doivent être parcourus pour localiser le fichier. Un chemin d'accès peut être relatif ou absolu.

Pour corriger cette erreur

- Vérifiez que le chemin d'accès inclut les éléments requis.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem, objet

Chemin d'accès introuvable

Path not found

Au cours d'une opération d'accès au fichier ou au disque, le système d'exploitation n'a pas été en mesure de trouver le chemin d'accès spécifié. Le chemin d'accès à un fichier comprend la spécification de lecteur plus les répertoires et sous-répertoires qui doivent être parcourus pour localiser le fichier. Un chemin d'accès peut être relatif ou absolu.

Pour corriger cette erreur

- Vérifiez et spécifiez à nouveau le chemin d'accès.

Voir aussi

Concepts

Types d'erreurs

Erreur dans le chemin d'accès

Path/File access error

Au cours d'une opération d'accès au fichier ou au disque, le système d'exploitation n'a pas pu établir de liaison entre le chemin d'accès et le nom de fichier.

Pour corriger cette erreur

1. Assurez-vous que la spécification de fichier est correctement mise en forme. Un nom de fichier peut contenir un chemin d'accès qualifié complet (absolu) ou relatif. Un chemin d'accès qualifié complet commence par le nom du lecteur (si le chemin d'accès se trouve sur un autre lecteur) et répertorie le chemin explicite de la racine au fichier. Tout chemin non complet est relatif au lecteur actif et au répertoire en cours.
2. Vérifiez que vous n'avez pas tenté d'enregistrer un fichier qui remplacerait un fichier existant en lecture seule. Si tel est le cas, modifiez l'attribut de lecture seule du fichier cible ou enregistrez le fichier sous un nom différent.
3. Vérifiez que vous n'avez pas tenté d'ouvrir un fichier en lecture seule en mode séquentiel **Output** ou **Append**. Si tel est le cas, ouvrez le fichier en mode **Input** ou modifiez l'attribut en lecture seule du fichier.
4. Vérifiez que vous n'avez pas tenté de modifier un projet Visual Basic à l'intérieur d'une base de données ou d'un document.

Voir aussi

Concepts

Types d'erreurs

Autorisation refusée (Visual Basic)

Permission denied (Visual Basic)

Une tentative d'écriture sur un disque protégé en écriture ou d'accès à un fichier verrouillé a été effectuée.

Pour corriger cette erreur

1. Pour ouvrir un fichier protégé en écriture, modifiez l'attribut de protection en écriture du fichier.
2. Assurez-vous qu'aucun autre processus n'a verrouillé le fichier et attendez avant d'ouvrir le fichier que l'autre processus le libère.
3. Pour accéder au Registre, vérifiez que vos autorisations d'utilisateur incluent bien ce type d'accès au Registre.

Voir aussi

Concepts

Types d'erreurs

Caractère non valide

Picture is not valid

Vous avez tenté d'assigner un format graphique autre qu'une bitmap, une icône ou un métafichier Windows à la propriété **Picture** d'un formulaire ou d'un contrôle.

Pour corriger cette erreur

- Vérifiez que le fichier que vous essayez de charger dans la propriété **Picture** est un fichier graphique valide pris en charge par Visual Basic.

Voir aussi

Concepts

Éléments graphiques pour les utilisateurs de Visual Basic 6.0

Erreur de l'imprimante

Printer error

Une erreur s'est produite au niveau de l'imprimante, mais aucune autre information n'a été retournée à l'ordinateur qui a envoyé le fichier.

Pour corriger cette erreur

- Examinez physiquement l'imprimante. Assurez-vous que toutes les connexions entre l'ordinateur et l'imprimante sont valides. La plupart des imprimantes fournissent un affichage des informations d'erreur, telles que « Hors ligne » ou « Plus de papier ».

Voir aussi

Concepts

Types d'erreurs

Argument ou appel de procédure non valide (Visual Basic)

Procedure call or argument is not valid (Visual Basic)

Une partie de l'appel ne peut pas être terminée.

Pour corriger cette erreur

- Vérifiez les plages autorisées pour les arguments afin de vous assurer qu'aucune disposition ne dépasse les valeurs autorisées.

Voir aussi

Concepts

Types d'erreurs

Le processus '<nomprocessus>' est introuvable

Process '<processname>' was not found

Le processus spécifié ne peut pas être localisé.

Pour corriger cette erreur

- Vérifiez que vous avez indiqué le nom de processus correct.

Voir aussi

Autres ressources

Gestion des processus

La propriété <NomPropriété> ne peut pas être une chaîne vide ou Nothing

Property <propertyname> cannot be set to an empty string or Nothing

La propriété requiert une valeur autre qu'une chaîne vide ("") ou **Nothing**. Il peut s'agir d'un résultat du calcul incorrect de la valeur.

Pour corriger cette erreur

- Fournissez une valeur valide.

Voir aussi

Référence

Nothing (Visual Basic)

Dépannage des exceptions : System.NullReferenceException

Concepts

Types d'erreurs

La propriété <NomPropriété> ne peut pas avoir la valeur Nothing

Property <propertyname> cannot be set to Nothing

La propriété requiert une valeur autre que **Nothing**. Cela peut être dû à un calcul inexact de la valeur.

Pour corriger cette erreur

- Entrez une valeur valide.

Voir aussi

Référence

Nothing (Visual Basic)

Dépannage des exceptions : System.NullReferenceException

Concepts

Types d'erreurs

L'index de tableau de propriétés est non valide

Property array index is not valid

La valeur fournie n'est pas valide pour un index de tableau de propriétés.

Pour corriger cette erreur

- Consultez la documentation du composant afin de vérifier que votre index se trouve dans la plage valide pour la propriété spécifiée.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

La procédure Property Let n'est pas définie et la procédure Property Get n'a pas retourné d'objet

Property let procedure not defined and property get procedure did not return an object

Certaines propriétés, méthodes et opérations ne peuvent s'appliquer qu'aux objets **Collection**. Vous avez spécifié une opération ou une propriété ne s'appliquant qu'aux collections, mais l'objet n'est pas une collection.

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom de l'objet ou de la propriété ou bien vérifiez que l'objet est un objet **Collection**.
2. Examinez la méthode **Add** utilisée pour ajouter l'objet à la collection afin d'être sûr que la syntaxe est correcte et que tous les identificateurs ont été orthographiés correctement.

Voir aussi

Référence

Collection, objet (Visual Basic)

Propriété introuvable

Property not found

Cet objet ne prend pas en charge la propriété spécifiée.

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom de la propriété.
2. Consultez la documentation de l'objet afin de vérifier que vous ne tentez pas d'accéder à une propriété, telle que "text", alors que l'objet prend en charge une propriété "caption" ou de nom similaire.

Voir aussi

Concepts

Types d'erreurs

Propriété ou méthode introuvable

Property or method not found

Une méthode d'objet ou une propriété d'objet référencée n'est pas définie.

Pour corriger cette erreur

- Vous avez peut-être mal orthographié le nom de l'objet. Pour vérifier quelles sont les propriétés et méthodes définies pour un objet, affichez l'Explorateur d'objets. Sélectionnez la bibliothèque d'objets appropriée afin d'afficher une liste des propriétés et méthodes disponibles.

Voir aussi

Concepts

Types d'erreurs

Valeur de propriété non valide

Property value is not valid

Une valeur en dehors de la plage autorisée a été affectée à l'une des propriétés d'un objet ou d'un contrôle.

Pour corriger cette erreur

- Remplacez la valeur de la propriété par un paramètre valide.

Voir aussi

Concepts

Types d'erreurs

Le membre public '<nommembre>' du type '<nomtype>' est introuvable

Public member '<membername>' on type '<typename>' not found

Le membre auquel vous tentez d'accéder est introuvable. Cette erreur peut se produire lors de l'accès à des membres non publics d'objets à liaison tardive.

Pour corriger cette erreur

- Si le membre existe, vérifiez qu'il est déclaré en tant que **Public**.

Voir aussi

Référence

Public (Visual Basic)

Concepts

Liaison anticipée et liaison tardive

'Redim' ne peut pas changer le nombre de dimensions

'ReDim' cannot change the number of dimensions

Une opération essaie d'utiliser l'instruction **ReDim** pour modifier le rang (nombre de dimensions) d'un tableau. **ReDim** peut modifier la taille d'une ou de plusieurs dimensions d'un tableau qui a déjà été déclaré formellement, mais il ne peut pas modifier le rang d'un tableau.

Pour corriger cette erreur

- Vérifiez que vous souhaitez modifier le rang d'un tableau au lieu des tailles de ses dimensions et, si possible, utilisez **Dim** pour déclarer un nouveau tableau avec le rang souhaité.

Voir aussi

Référence

ReDim, instruction (Visual Basic)
Dim, instruction (Visual Basic)

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic

'ReDim' ne peut changer que la dimension la plus à droite

'ReDim' can only change the right-most dimension

Une instruction **ReDim** a tenté d'utiliser le mot clé **Preserve** pour modifier une dimension d'un tableau autre que la dernière dimension. Lorsque vous utilisez **Preserve**, vous ne pouvez redimensionner que la dernière dimension d'un tableau. Pour toutes les autres dimensions, vous devez spécifier la même taille par rapport au tableau existant.

Pour corriger cette erreur

- Supprimez le mot clé **Preserve**.

Voir aussi

Référence

ReDim, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Preserve

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic

L'opérande 'ReDim Preserve' ne peut pas avoir la valeur Nothing

'ReDim' Preserve operand cannot be Nothing

Une instruction **ReDim** tente d'utiliser le mot clé **Preserve** pour modifier une dimension d'un tableau autre que la dernière dimension, mais ne fournit aucune valeur valide pour son opérande.

Pour corriger cette erreur

- Remplacez l'opérande **Preserve** par une valeur valide.

Voir aussi

Référence

ReDim, instruction (Visual Basic)
Dim, instruction (Visual Basic)
Preserve

Concepts

Vue d'ensemble des tableaux dans Visual Basic
Tableaux multidimensionnels dans Visual Basic

Impossible de trouver la clé de Registre '<nomclé>'

Registry key '<keyname>' could not be found

Une clé de Registre est introuvable.

Pour corriger cette erreur

1. Vérifiez l'orthographe du nom de la clé.
2. Assurez-vous que vous disposez des autorisations de sécurité suffisantes pour accéder à la clé.

Voir aussi

Référence

Liste des registres

Concepts

Accès au Registre pour les utilisateurs de Visual Basic 6.0

Impossible de créer la clé de Registre '<nomclé>'

Registry key '<keyname>' could not be created

La clé de Registre spécifiée n'a pas pu être créée, probablement en raison d'autorisations de sécurité insuffisantes.

Pour corriger cette erreur

- Vérifiez que vous disposez des autorisations de sécurité suffisantes pour accéder à la clé.

Voir aussi

Référence

Liste des registres

Concepts

Accès au Registre pour les utilisateurs de Visual Basic 6.0

Remplacements trop longs

Replacements too long

Vous avez spécifié un remplacement qui dépassait la longueur autorisée.

Pour corriger cette erreur

- Consultez la documentation du composant pour connaître la restriction de longueur.

Voir aussi

Tâches

Comment : rechercher de façon interactive

Reprendre sans gestion d'erreur

Resume without error

Une instruction **Resume** est apparue en dehors du code de gestion des erreurs ou le code est passé dans un gestionnaire d'erreurs bien qu'il n'y ait pas eu d'erreur.

Pour corriger cette erreur

1. Déplacez l'instruction **Resume** dans un gestionnaire d'erreurs ou supprimez-la.
2. Les passages à des étiquettes sont impossibles à travers les procédures. Par conséquent, cherchez dans la procédure l'étiquette qui identifie le gestionnaire d'erreurs. Si vous trouvez une étiquette dupliquée spécifiée comme cible d'une instruction **GoTo** et qui n'est pas une instruction **On Error GoTo**, changez l'étiquette de ligne de manière à ce qu'elle corresponde à la cible attendue.

Voir aussi

Concepts

Vue d'ensemble de la gestion non structurée des exceptions

Le dossier racine ne peut pas être renommé

Root folder cannot be renamed

Une tentative a été effectuée pour renommer le répertoire racine. Cela peut être dû à une analyse incorrecte du chemin d'accès.

Pour corriger cette erreur

- Vérifiez que le chemin d'accès est correct.

Voir aussi

Tâches

Comment : renommer un répertoire dans Visual Basic

Comment : déplacer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.RenameDirectory, méthode

My.Computer.FileSystem.RenameFile, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Le texte recherché est introuvable

Search text not found

Le texte que vous avez spécifié n'a pas été trouvé.

Pour corriger cette erreur

- Vérifiez que le texte spécifié est correctement orthographié.

Voir aussi

Tâches

Comment : rechercher de façon interactive

Set non pris en charge au moment de l'exécution

Set not supported at run time

Vous avez tenté de définir ou de modifier une propriété dont la valeur peut seulement être définie au moment du design.

Pour corriger cette erreur

1. Supprimez la référence à la propriété de votre code.
2. Modifiez la référence afin de retourner uniquement la valeur de la propriété au moment de l'exécution.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Set non pris en charge (propriété en lecture seule)

Set not supported (read-only property)

Vous avez tenté de définir ou de modifier une propriété en lecture seule.

Pour corriger cette erreur

1. Supprimez la référence à la propriété de votre code.
2. Modifiez la référence afin de retourner uniquement la valeur de la propriété au moment de l'exécution.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Set non autorisé

Set not permitted

Vous avez tenté de modifier une propriété dont les paramètres ne peuvent pas être définis au moment de l'exécution ou peuvent uniquement être définis sous certaines conditions. Par exemple, vous avez peut-être essayé de modifier les paramètres des propriétés **Appearance**, **ControlBox**, **MinButton** ou **MaxButton** du formulaire au moment de l'exécution, ou vous avez peut-être essayé d'affecter à la propriété **Visible** la valeur **False** du dernier sous-menu visible d'un menu parent.

Pour corriger cette erreur

- Vérifiez la propriété et déterminez les conditions sous lesquelles elle peut être définie.

Voir aussi

Tâches

Comment : modifier des propriétés de projet et des paramètres de configuration

Impossible de combiner SimplifiedChinese et VbStrConv.TraditionalChinese

SimplifiedChinese and VbStrConv.TraditionalChinese cannot be combined

Votre application essaie de combiner les membres d'énumération **VbStrConv SimplifiedChinese** et **TraditionalChinese**, qui s'excluent mutuellement.

Pour corriger cette erreur

- Supprimez **VbStrConv.SimplifiedChinese** ou **VbStrConv.TraditionalChinese**.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Certains fichiers et dossiers ont provoqué des exceptions lors de l'opération

Some files and folders caused exceptions during the operation

Des répertoires et des fichiers peuvent générer des exceptions lorsqu'une opération est tentée alors que l'utilisateur ne dispose des autorisations nécessaires.

Pour corriger cette erreur

- Ajoutez les autorisations nécessaires pour accéder à tous les fichiers et dossiers.

Voir aussi

Tâches

Dépannage : lecture et écriture dans des fichiers texte

Référence

FileIOPermission

Certaines sous-clés ne peuvent pas être supprimées

Some subkeys cannot be deleted

Une tentative a été effectuée pour supprimer une clé de Registre, mais l'opération a échoué car certaines sous-clés ne peuvent pas être supprimées. Cela est généralement dû à une absence d'autorisations.

Pour corriger cette erreur

- Assurez-vous que vous disposez des autorisations nécessaires pour supprimer les sous-clés spécifiées.

Voir aussi

Référence

My.Computer.Registry, objet
DeleteSubKey
DeleteSubKey
RegistryPermission

Le dossier source et le dossier cible sont identiques

Source folder and target folder are the same

Une tentative a été faite pour copier ou déplacer un fichier, mais le répertoire source et le répertoire cible spécifiés sont identiques. Cela peut être dû à une analyse incorrecte du chemin d'accès.

Pour corriger cette erreur

- Vérifiez que vous avez fourni les répertoires source et cible corrects.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic
Comment : déplacer un fichier dans Visual Basic
Comment : copier des fichiers avec un modèle spécifique dans un répertoire dans Visual Basic

Référence

My.Computer.FileSystem.MoveFile, méthode
My.Computer.FileSystem.CopyFile, méthode

Le nom de source spécifié dans EventLogSource est enregistré dans un journal autre que celui spécifié dans EventLogName

Source name specified in EventLogSource is registered to a log other than that specified in EventLogName

EventLog essaie de faire référence à une source enregistrée dans un autre journal. Si vous écrivez des entrées dans un journal des événements, vous devez spécifier la propriété Source. La propriété **Source** inscrit votre composant en tant que source d'entrées valide dans le journal des événements. Une seule source peut être associée à (et donc écrire des entrées dans) un seul journal des événements à la fois.

Par défaut, si vous tentez d'écrire une entrée sans avoir au préalable inscrit le composant en tant que source valide, le système inscrit automatiquement la source dans le journal des événements, en utilisant la valeur de la propriété **Source** comme chaîne source.

Pour corriger cette erreur

- Assurez-vous que la source est enregistrée dans le journal correct. Pour ce faire, utilisez la méthode CreateEventSource ou l'une de ses surcharges pour spécifier une chaîne qui identifie uniquement votre composant dans le journal des événements.

Voir aussi

Tâches

Comment : ajouter votre application en tant que source d'entrées d'un journal des événements

Comment : supprimer une source d'événements

Concepts

Référencement des journaux des événements

Autres ressources

Gestion des journaux des événements

Fonction DLL spécifiée introuvable

Specified DLL function not found

La bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) d'une référence de bibliothèque utilisateur a été trouvée, mais la fonction DLL spécifiée est introuvable dans la DLL.

Pour corriger cette erreur

1. Spécifiez un nombre valide dans la déclaration de fonction.
2. Vérifiez que le nom de la DLL et l'alias sont corrects.

Voir aussi

Concepts

Types d'erreurs

Autres ressources

Support technique et accessibilité

Le journal des événements spécifié n'existe pas sur cet ordinateur

Specified event log does not exist on this machine

Une tentative a été effectuée pour accéder à un journal des événements qui n'existe pas sur cet ordinateur.

Pour corriger cette erreur

- Vérifiez que le nom et le chemin d'accès du fichier fournis sont corrects.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Autres ressources

Gestion des journaux des événements

La clé de Registre spécifiée n'existe pas

Specified registry key does not exist

La clé de Registre spécifiée n'existe pas.

Pour corriger cette erreur

- Vérifiez la clé de Registre fournie pour vous assurer que vous avez correctement spécifié le chemin d'accès.
- Créez la clé de Registre avant toute tentative d'écriture.

Voir aussi

Tâches

Procédure pas à pas : création d'une clé de Registre et modification de ses valeurs

Référence

My.Computer.Registry, objet

RegistryKey

Concepts

Tâches courantes relatives au Registre

La clé de Registre spécifiée n'est pas valide car elle contient au moins deux barres obliques inverses consécutives

Specified registry key is not valid because it contains two or more consecutive backslashes

Une clé de Registre spécifiée avec un chemin d'accès contient deux ou plusieurs barres obliques inverses consécutives. Cela peut être dû à la combinaison de plusieurs chaînes pour former le chemin d'accès et à l'insertion par inadvertance d'un trop grand nombre de barres obliques inverses.

Pour corriger cette erreur

- Examinez la clé de Registre spécifiée pour déterminer l'emplacement et la raison pour laquelle des barres obliques inverses supplémentaires ont été insérées.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.Registry, objet

Le chemin d'accès du Registre spécifié ne commence pas par un nom de ruche valide

Specified registry path does not start with a valid hive name

Le chemin d'accès du Registre spécifié ne commence pas par un nom de ruche valide. Les noms de ruches valides sont :

- **HKEY_CLASSES_ROOT**
- **HKEY_CURRENT_CONFIG**
- **HKEY_CURRENT_USER**
- **HKEY_DYNDATA**
- **HKEY_LOCAL_MACHINE**
- **HKEY_PERFORMANCE_DATA**
- **HKEY_USERS**

Pour corriger cette erreur

- Vérifiez qu'un nom de ruche valide a été fourni.

Voir aussi

Référence

My.Computer.Registry, objet

Concepts

Tâches courantes relatives au Registre

Autres ressources

Manipulation des chaînes

Instruction Stop rencontrée

Stop statement encountered

Une instruction **Stop**, qui interrompt l'exécution, a été rencontrée.

Pour corriger cette erreur

- Supprimez l'instruction **Stop**.

Voir aussi

Référence

Stop, instruction (Visual Basic)

Liste des flux de contrôle

Concepts

Instructions Stop en Visual Basic

'StrConv.LinguisticCasing' nécessite 'StrConv.LowerCase' ou 'StrConv.UpperCase'

'StrConv.LinguisticCasing' requires 'StrConv.LowerCase' or 'StrConv.UpperCase'

Vous avez tenté d'utiliser **StrConv.LinguisticCasing**, qui est seulement valide dans une combinaison avec **StrConv.LowerCase** ou **StrConv.UpperCase**.

Pour corriger cette erreur

- Utilisez **StrConv.LowerCase** ou **StrConv.UpperCase** conjointement avec **StrConv.LinguisticCasing**.

Voir aussi

Référence

StrConv, fonction

Constantes StrConv pour les utilisateurs de Visual Basic 6.0

La longueur de chaîne dépasse la longueur maximale (32 767 caractères) autorisée pour les API 'FileSystem'

String length exceeds maximum length of 32767 characters for 'FileSystem' APIs

La longueur d'une chaîne dépasse la longueur maximale qui est égale à 32 767 caractères.

Pour corriger cette erreur

- Raccourcissez la chaîne.

Voir aussi

Autres ressources

Chaînes en Visual Basic

Sub ou Function non défini (Visual Basic)

Sub or Function not defined (Visual Basic)

Un **Sub** ou un **Function** doit être défini afin d'être appelé. Cette erreur peut avoir plusieurs causes :

- Faute d'orthographe dans le nom de la procédure.
- Tentative d'appeler une procédure à partir d'un autre projet sans ajouter explicitement de référence à ce projet dans la boîte de dialogue **Références**.
- Spécification d'une procédure qui n'est pas visible pour la procédure appelante.
- Déclaration d'une routine de bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) Windows ou d'une routine de ressource de code Macintosh qui ne se trouve pas dans la bibliothèque spécifiée ou dans la ressource de code.

Pour corriger cette erreur

1. Assurez-vous que le nom de la procédure est correctement orthographié.
2. Recherchez le nom du projet contenant la procédure que vous voulez appeler dans la boîte de dialogue **Références**. Si celui-ci n'apparaît pas, cliquez sur le bouton **Parcourir** pour le rechercher. Activez la case à cocher située à gauche du nom de projet, puis cliquez sur **OK**.
3. Vérifiez le nom de la routine.

Voir aussi

Référence

Sub, instruction (Visual Basic)

Function, instruction (Visual Basic)

Concepts

Types d'erreurs

Références de projet

Indice hors limites (Visual Basic)

Subscript out of range (Visual Basic)

Un indice de tableau n'est pas valide car il se situe en dehors de la plage autorisée. La plus petite valeur d'indice d'une dimension correspond toujours à 0 et la plus grande valeur d'indice est retournée par la méthode **GetUpperBound** de cette dimension.

Pour corriger cette erreur

- Modifiez l'indice de sorte qu'il se situe dans la plage valide.

Voir aussi

Référence

System.Array.GetUpperBound(System.Int32)

Concepts

Vue d'ensemble des tableaux dans Visual Basic

Le journal des événements système ne peut pas être supprimé

System event log cannot be deleted

Une tentative a été effectuée pour supprimer le journal des événements système, qui ne peut pas être supprimé. Le journal système assure le suivi des événements système, par exemple le démarrage du système et les défaillances matérielles.

Pour corriger cette erreur

- Il est préférable que votre d'application écrive dans un journal d'application ou personnalisé, au lieu du journal des événements système.
- Ne tentez pas de supprimer le journal des événements système.

Voir aussi

Tâches

Comment : créer et supprimer des journaux des événements personnalisés

Autres ressources

Gestion des journaux des événements

Le dossier cible est un fichier

Target folder is a file

Vous spécifiez un fichier tout en essayant d'exécuter une action relative à un dossier.

Pour corriger cette erreur

- Vérifiez le chemin d'accès que vous fournissez pour vous assurer que vous spécifiez un dossier plutôt qu'un fichier.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Comment : déterminer le chemin d'accès absolu d'un fichier dans Visual Basic

Référence

My.Computer.FileSystem, objet

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Propriétés des fichiers, répertoires et lecteurs dans Visual Basic

TargetFilePath spécifie un dossier existant

TargetFilePath specifies an existing folder

Le paramètre *TargetFilePath* spécifie un répertoire existant, mais doit spécifier le nouveau nom du répertoire.

Pour corriger cette erreur

- Modifiez *TargetFilePath* pour spécifier un nouveau répertoire plutôt qu'un répertoire déjà existant.

Voir aussi

Tâches

Comment : déplacer un fichier dans Visual Basic
Comment : déplacer un répertoire dans Visual Basic
Comment : renommer un fichier dans Visual Basic
Comment : renommer un répertoire dans Visual Basic

TextFieldParser ne prend pas en charge les jetons de commentaires qui contiennent un blanc

TextFieldParser does not support comment tokens that contain whitespace

Un jeton de commentaire contenant un espace blanc a été fourni. **TextFieldParser** ne prend pas en charge les jetons de commentaires qui contiennent un espace blanc à moins que l'espace blanc soit situé au début du jeton. Un espace blanc présent au début d'un jeton est ignoré.

Pour corriger cette erreur

- Fournissez un jeton de commentaire correct.

Voir aussi

Référence

TextFieldParser.CommentTokens, propriété

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser ne prend pas en charge les délimiteurs qui contiennent des caractères de fin de ligne

TextFieldParser does not support delimiters that contain endline characters

Un délimiteur qui contient un caractère de fin de ligne a été fourni, mais **TextFieldParse** ne prend pas en charge ce type de délimiteur.

Pour corriger cette erreur

- Entrez un délimiteur correct.

Voir aussi

Référence

TextFieldParser.Delimiters, propriété
TextFieldParser.SetDelimiters, méthode
TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

TextFieldParser ne peut pas achever l'opération de lecture, car la taille maximale de la mémoire tampon a été dépassée

TextFieldParser is unable to complete the read operation because maximum buffer size has been exceeded

L'opération ne peut pas être achevée car la taille maximale de la mémoire tampon (10 000 000 octets) a été dépassée.

Pour corriger cette erreur

- Assurez-vous que le fichier ne contient pas de champs incorrects.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem.OpenTextFieldParser, méthode

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

L'adresse de UploadFile doit inclure un nom de fichier

The address for UploadFile needs to include a filename

Lorsque vous téléchargez des fichiers à l'aide de la méthode **UploadFile**, vous devez fournir un nom de fichier dans le chemin d'accès de destination.

Pour corriger cette erreur

- Entrez un nom de fichier.

Voir aussi

Tâches

Comment : transférer un fichier dans Visual Basic

Référence

My.Computer.Network, objet

My.Computer.Network.UploadFile, méthode

Autres ressources

Opérations de réseau dans le .NET Framework avec Visual Basic

Le fichier est déjà ouvert

The file is already open

Vous avez essayé d'ouvrir un fichier qui l'est déjà.

Pour corriger cette erreur

- Fermez le fichier avant d'essayer de l'ouvrir.

  - ou -

- Supprimez l'instruction **FileOpen** redondante de votre code.

Voir aussi

Référence

FileOpen, fonction

My.Computer.FileSystem, objet

Le fichier est actuellement fermé

The file is currently closed

Une tentative a été effectuée pour exécuter une opération qui peut être exécutée uniquement sur un fichier ouvert, mais le fichier est actuellement fermé.

Pour corriger cette erreur

- Ouvrez le fichier et recommencez l'opération.

Voir aussi

Référence

FileOpen, fonction
FileClose, fonction
My.Computer.FileSystem, objet

Autres ressources

Accès au fichier avec Visual Basic

Le fichier est actuellement ouvert en lecture

The file is currently open for reading

Une tentative a été effectuée pour accéder à un fichier qui est actuellement ouvert en lecture.

Pour corriger cette erreur

- Fermez le fichier et recommencez l'opération.

Voir aussi

Tâches

Comment : obtenir des informations à propos d'un fichier dans Visual Basic
Comment : lire des fichiers texte dans Visual Basic
Comment : lire des fichiers binaires dans Visual Basic

Référence

FileIOPermission

Le fichier est actuellement ouvert en écriture

The file is currently open for writing

Une tentative a été effectuée pour accéder à un fichier qui est actuellement ouvert en écriture.

Pour corriger cette erreur

- Fermez le fichier et recommencez l'opération.

Voir aussi

Tâches

Comment : obtenir des informations à propos d'un fichier dans Visual Basic
Comment : insérer du texte dans des fichiers dans Visual Basic
Comment : écrire dans des fichiers binaires dans Visual Basic

Référence

FileIOPermission

Le dossier ne peut pas être créé car un fichier existe déjà avec le même chemin d'accès

The folder cannot be created since a file already exists with the same path

Une tentative a été effectuée pour créer un répertoire, mais il existe déjà un fichier avec ce chemin d'accès. Cela peut être dû à une analyse incorrecte des chemins d'accès, notamment lorsque vous utilisez des chemins d'accès relatifs.

Pour corriger cette erreur

- Vérifiez que vous entrez le chemin d'accès correct.
- Supprimez le fichier redondant s'il n'est plus nécessaire.

Voir aussi

Tâches

Comment : créer un répertoire dans Visual Basic
Comment : renommer un répertoire dans Visual Basic

Référence

My.Computer.FileSystem, objet
My.Computer.FileSystem.CreateDirectory, méthode
My.Computer.FileSystem.RenameDirectory, méthode

Autres ressources

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Le chemin d'entrée fait référence à un fichier mais se termine par un caractère de séparation de répertoire

The input path refers to a file but ends with a directory separator character

Une opération qui requiert un nom de fichier a été tentée, mais le nom fourni se termine par un caractère de séparation de répertoire (\). Cela peut être dû à une analyse incorrecte du chemin d'accès.

Pour corriger cette erreur

- Vérifiez que le chemin d'accès fourni est correct.
- Supprimez le caractère superflu.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Autres ressources

Lecture à partir de fichiers en Visual Basic

Écriture dans des fichiers en Visual Basic

Le chemin d'accès n'est pas défini

The path has not been set

Une opération sur un fichier, par exemple le déplacement ou la copie d'un fichier, a été tentée sans fournir le chemin d'accès nécessaire. Cela peut être dû à une analyse incorrecte du chemin d'accès.

Pour corriger cette erreur

- Vérifiez que le chemin d'accès a été fourni et qu'il est correct.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Autres ressources

Accès au fichier avec Visual Basic

Le serveur distant n'existe pas ou n'est pas disponible (Visual Basic)

The remote server machine does not exist or is unavailable (Visual Basic)

Vous tentez de vous connecter à un serveur distant qui est indisponible ou qui n'existe pas. Cette erreur peut provenir d'une connexion perdue au réseau.

Pour corriger cette erreur

- Reconnectez-vous au réseau et recommencez.

Voir aussi

Concepts

Types d'erreurs

Le dossier source n'existe pas

The source folder does not exist

Vous avez essayé une opération qui référence un dossier source qui n'existe pas. Cela peut être dû à un chemin d'accès incorrect.

Pour corriger cette erreur

- Vérifiez que vous utilisez le chemin d'accès correct, en particulier si vous utilisez des chemins d'accès relatifs.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Autres ressources

Lecture à partir de fichiers en Visual Basic

Création, suppression et déplacement de fichiers et de répertoires en Visual Basic

Le chemin d'accès spécifié n'existe pas

The specified path does not exist

Un chemin d'accès spécifié n'existe pas, en raison peut-être d'une erreur associant des chaînes.

Pour corriger cette erreur

- Assurez-vous que le chemin d'accès est correct.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Concepts

Conversion entre des chaînes et d'autres types

Autres ressources

Analyse de chaînes en Visual Basic

Le flux passé à TextFieldParser est illisible

The stream passed to TextFieldParser cannot be read

L'objet TextFieldParser ne peut pas lire le flux de données qui lui a été transmis. Cela peut être dû à la tentative de lire un fichier qui n'est pas un fichier texte.

Pour corriger cette erreur

- Vérifiez que le fichier est un fichier texte.

Voir aussi

Tâches

Comment : lire des fichiers texte avec plusieurs formats dans Visual Basic

Référence

My.Computer.FileSystem.OpenTextFieldParser, méthode

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

La valeur de <NomArgument> doit être un nombre positif

The value of <argumentname> must be a positive number

La valeur de la propriété ReserveDiskSpace doit être supérieure à zéro.

La propriété **ReserveDiskSpace** spécifie, en octets, la quantité d'espace disque libre nécessaire pour que les messages puissent être écrits dans le fichier journal.

Pour corriger cette erreur

- Affectez une valeur positive à la propriété **ReserveDiskSpace**.

Voir aussi

Référence

My.Application.Log, objet
My.Log, objet
ReserveDiskSpace

La valeur de <NomArgument> doit être supérieure ou égale à 1 000

The value of <argumentname> must be greater than or equal to 1000

La valeur de la propriété MaxFileSize doit être supérieure ou égale à 1 000.

La propriété **MaxFileSize** spécifie le nombre maximum d'octets autorisé dans un fichier journal avant qu'un nouveau fichier journal soit créé.

Pour corriger cette erreur

- Affectez à la propriété **MaxFileSize** une valeur supérieure ou égale à 1 000.

Voir aussi

Référence

My.Application.Log, objet
My.Log, objet
MaxFileSize

Ce tableau est prédéfini ou est temporairement verrouillé (Visual Basic)

This array is fixed or temporarily locked (Visual Basic)

Les causes de cette erreur peuvent être les suivantes :

- Utilisation de **ReDim** pour modifier le nombre d'éléments d'un tableau de taille fixe.
- Redimensionnement d'un tableau dynamique au niveau du module, dans lequel un élément a été passé comme argument à une procédure. Si un élément est passé, le tableau est verrouillé afin d'empêcher la désallocation de mémoire pour le paramètre de référence dans la procédure.
- Tentative d'assignation d'une valeur à une variable **Variant** contenant un tableau, alors que la variable **Variant** est actuellement verrouillée.

Pour corriger cette erreur

1. Rendez le tableau d'origine dynamique et non fixe en le déclarant avec **ReDim** (si le tableau est déclaré dans une procédure) ou sans spécifier le nombre d'éléments (si le tableau est déclaré au niveau du module).
2. Déterminez si vous devez réellement passer l'élément, car il est visible dans toutes les procédures du module.
3. Déterminez l'élément qui verrouille la variable **Variant** et remédiez au problème.

Voir aussi

Autres ressources

Tableaux dans Visual Basic

Cette clé est déjà associée à un élément de cette collection

This key is already associated with an element of this collection

La clé spécifiée pour un membre d'une collection identifie déjà un autre membre de la collection. Une clé est une chaîne spécifiée dans la méthode **Add** qui identifie de façon unique un membre spécifique d'une collection.

Pour corriger cette erreur

- Utilisez une clé différente pour ce membre.

Voir aussi

Concepts

Types d'erreurs

Cette opération ne peut avoir lieu que lorsque le fichier est fermé.

This operation can only be done when the File is closed

Lorsqu'un fichier est ouvert, une tentative a été effectuée pour exécuter une opération qui peut avoir lieu uniquement lorsque le fichier est fermé.

Pour corriger cette erreur

- Fermez le fichier et recommencez l'opération.

Voir aussi

Tâches

Comment : obtenir des informations à propos d'un fichier dans Visual Basic

Référence

FileIOPermission

FileClose

Impossible de connecter cette application à instance unique à l'instance d'origine

This single-instance application could not connect to the original instance

Cette application à instance unique ne peut pas être connectée à l'instance d'origine. Ce problème peut avoir plusieurs causes :

- L'instance d'origine a cessé de répondre.
- L'application n'a pas l'autorisation de créer des objets de noyau. Pour plus d'informations sur les objets du noyau, consultez Mutex.

  Le nom de base des objets du noyau provient de la concaténation du GUID, du numéro de version principal et du numéro de version secondaire de l'assembly. Par exemple, le nom de base peut être 3639f15d-9547-43da-8145-60da347829915.1.

Pour corriger cette erreur lors du développement de l'application

1. Vérifiez que l'application n'est pas bloquée.
2. Vérifiez que l'application dispose des autorisations nécessaires pour créer des objets de noyau.
3. Redémarrez l'instance d'origine de l'application.
4. Redémarrez l'ordinateur pour effacer les processus qui utilisent la ressource requise pour se connecter à l'application de l'instance d'origine.
5. Prenez note des circonstances dans lesquelles l'erreur est survenue et appelez les services de Support Technique de Microsoft.

Voir aussi

Tâches

Comment : spécifier le comportement d'instanciation pour une application

Autres ressources

Présentation du débogueur

Support technique et accessibilité

Ce système n'offre pas de prise en charge des paramètres régionaux pour le japonais

This system does not contain support for the Japanese locale.

Vous essayez d'utiliser le membre d'énumération **VbStrConv.Japanese**, mais votre système ne prend pas en charge les paramètres régionaux pour le japonais.

Pour corriger cette erreur

- Procédez à une mise à niveau de l'ordinateur afin d'assurer la prise en charge des paramètres régionaux pour le japonais ou exécutez l'application sur un ordinateur qui prend en charge ces paramètres régionaux.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Ce système n'offre pas de prise en charge des paramètres régionaux spécifiés

This system does not contain support for the Locale specified.

Vous essayez d'utiliser l'énumération **VbStrConv**, mais votre système ne prend pas en charge les paramètres régionaux spécifiés.

Pour corriger cette erreur

- Procédez à une mise à niveau de l'ordinateur afin d'assurer la prise en charge des paramètres régionaux ou exécutez l'application sur un ordinateur qui prend en charge ces paramètres régionaux.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Ce système n'offre pas de prise en charge des paramètres régionaux pour le chinois simplifié

This system does not contain support for the SimplifiedChinese locale

Vous essayez d'utiliser le membre d'énumération **VbStrConv.SimplifiedChinese**, mais votre système ne prend pas en charge les paramètres régionaux pour le chinois simplifié.

Pour corriger cette erreur

- Procédez à une mise à niveau de l'ordinateur afin d'assurer la prise en charge des paramètres régionaux pour le chinois simplifié ou exécutez l'application sur un ordinateur qui prend en charge ces paramètres régionaux.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Ce système n'offre pas de prise en charge des paramètres régionaux pour le chinois traditionnel

This system does not contain support for the Traditional Chinese locale

Vous avez essayé d'utiliser le membre d'énumération **VbStrConv.TraditionalChinese**, mais votre système ne prend pas en charge les paramètres régionaux pour le chinois traditionnel.

Pour corriger cette erreur

- Procédez à une mise à niveau de l'ordinateur afin d'assurer la prise en charge des paramètres régionaux pour le chinois traditionnel ou exécutez l'application sur un ordinateur qui prend en charge ces paramètres régionaux.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Trop de clients pour cette DLL

Too many DLL application clients

La bibliothèque de liens dynamiques (DLL, Dynamic-Link Library) de Visual Basic autorise seulement l'accès d'un nombre limité d'applications hôtes. Votre application et d'autres applications qui sont des hôtes Visual Basic (certaines sont accessibles par votre application) tentent d'accéder simultanément à la DLL Visual Basic.

Pour corriger cette erreur

- Réduisez le nombre d'applications ouvertes qui accèdent à Visual Basic.

Voir aussi

Concepts

Types d'erreurs

Trop de fichiers

Too many files

Plus de fichiers que ceux qui sont autorisés par le système d'exploitation ont été créés dans le répertoire racine ou plus de fichiers que le nombre spécifié dans le paramètre **files=** de votre fichier CONFIG.SYS ont été ouverts.

Pour corriger cette erreur

1. Si votre programme ouvre, ferme ou enregistre des fichiers dans le répertoire racine, modifiez votre programme afin qu'il utilise un sous-répertoire.
2. Augmentez le nombre de fichiers spécifié dans le paramètre **files=** de votre fichier CONFIG.SYS et redémarrez votre ordinateur.

Voir aussi

Concepts

Types d'erreurs

Incompatibilité de type (Visual Basic)

Type mismatch (Visual Basic)

Vous avez tenté de convertir une valeur en un autre type d'une façon non valide.

Pour corriger cette erreur

1. Vérifiez que l'assignation est valide.
2. Vérifiez que vous n'avez pas passé un objet à une procédure qui attend une seule propriété ou valeur.
3. Vérifiez que vous n'avez pas utilisé un nom de module ou de projet lorsqu'une expression était attendue.

Voir aussi

Concepts

Types d'erreurs

Le type de l'argument '<NomArgument>' est '<NomType>', qui n'est pas numérique

Type of argument '<argumentname>' is '<typename>', which is not numeric

Un argument non numérique se produit lorsqu'un argument numérique est attendu.

Pour corriger cette erreur

- Vérifiez que l'argument est numérique.

Voir aussi

Concepts

Passage d'arguments par valeur et par référence
Mécanisme de passage de paramètres pour les utilisateurs de Visual Basic 6.0

Impossible d'obtenir le nom des ports série en raison d'une erreur système interne

Unable to get serial port names because of an internal system error

Une erreur interne s'est produite durant l'appel de la propriété **My.Computer.Ports.SerialPortNames**.

Pour corriger cette erreur

1. Pour obtenir des informations de dépannage, consultez Présentation du débogueur.
2. Prenez note des circonstances dans lesquelles l'erreur est survenue et appelez les services de Support Technique de Microsoft.

Voir aussi

Référence

My.Computer.Ports.SerialPortNames, propriété

Autres ressources

Présentation du débogueur

Support technique et accessibilité

Impossible d'obtenir un flux pour le journal

Message d'erreur
Impossible d'obtenir un flux pour le journal. Des noms de fichiers éventuels fondés sur <nom> sont déjà utilisés.
Unable to obtain a stream for the log. Potential file names based on <name> are already in use.

La classe FileLogTraceListener ne peut pas créer un nouveau fichier journal parce que tous les noms de fichiers journaux éventuels fondés sur <nom> sont déjà utilisés.

La présence d'un trop grand nombre de fichiers journaux peut indiquer un problème architectural avec l'application. Pour plus d'informations, consultez la documentation de la classe **FileLogTraceListener**.

Pour corriger cette erreur

1. Affectez Daily ou Weekly à la propriété LogFileCreationSchedule pour inclure un horodateur dans le nom de fichier journal.
2. Archivez les journaux existants et supprimez-les de l'ordinateur pour permettre à l'objet **FileLogTraceListener** de créer de nouveaux journaux.

Voir aussi

Référence
My.Application.Log, objet
My.Log, objet
FileLogTraceListener
LogFileCreationSchedule

Impossible de lire les champs de largeur fixe, car FieldWidths a la valeur Nothing ou est vide

Unable to read fixed width fields because FieldWidths is Nothing or empty

TextFieldParser ne peut pas lire les champs à largeur fixe car la propriété **FieldWidths** a la valeur **Nothing** ou est vide.

Pour corriger cette erreur

- Affectez une valeur valide à **FieldWidths**.

Voir aussi

Tâches

Comment : lire des fichiers texte de largeur fixe dans Visual Basic

Référence

TextFieldParser.SetFieldWidths, méthode

TextFieldParser.FieldWidths, propriété

TextFieldParser, objet

Impossible de lire les champs délimités, car Delimiters a la valeur Nothing ou est vide

Unable to read delimited fields because Delimiters is Nothing or empty

TextFieldParser ne peut pas lire le fichier car la propriété **Delimiters** a la valeur **Nothing** ou est vide.

Pour corriger cette erreur

- Entrez une valeur valide pour **Delimiters**.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Référence

TextFieldParser.SetDelimiters, méthode

TextFieldParser.Delimiters, propriété

TextFieldParser, objet

Concepts

Analyse des fichiers texte avec l'objet TextFieldParser

Les guillemets doubles ne constituent pas un jeton de commentaire valide pour les champs délimités lorsque EscapeQuote a la valeur True

Unable to read delimited fields because a double quote is not a legal delimiter when EscapeQuotes is set to True

TextFieldParser ne peut pas lire le fichier car un guillemet (") a été fourni comme délimiteur et **EscapeQuotes** a la valeur **True**.

Pour corriger cette erreur

- Affectez à **EscapeQuotes** la valeur **False**.

Voir aussi

Tâches

Comment : lire des fichiers texte délimités par des virgules dans Visual Basic

Référence

TextFieldParser.SetDelimiters, méthode

TextFieldParser.Delimiters, propriété

TextFieldParser, objet

Impossible d'exécuter la commande PING, car une connexion réseau n'est pas disponible

Unable to ping because a network connection is not available

L'opération ne peut pas être exécutée car aucune connexion réseau n'est disponible.

Pour corriger cette erreur

- Connectez-vous au réseau et recommencez l'opération.

Voir aussi

Référence

My.Computer.Network.Ping, méthode
My.Computer.Network.IsAvailable, propriété
My.Computer.Network, objet

Autres ressources

Opérations de réseau dans le .NET Framework avec Visual Basic

Impossible de recevoir des événements de l'objet, car le nombre maximal de récepteurs d'événements pris en charge est déjà atteint

Unable to sink events of object because the object is already firing events to the maximum number of event receivers it supports

Le nombre maximal de récepteurs d'événements pris en charge par l'objet a été dépassé.

Pour corriger cette erreur

- Réduisez le nombre de récepteurs d'événements.

Voir aussi

Tâches

Comment : déclencher des événements gérés par un récepteur COM
Procédure pas à pas : gestion des événements

Impossible d'écrire dans le fichier journal, car cette opération entraînerait un dépassement de la valeur MaximumSize

Unable to write to log file because writing to it would cause it to exceed MaximumSize value

La classe FileLogTraceListener n'a pas pu écrire dans le fichier journal pour les raisons suivantes :

- La taille du fichier journal (en octets) est supérieure à la valeur de la propriété MaxFileSize

  — et —

- La valeur de la propriété DiskSpaceExhaustedBehavior est ThrowException.

Pour corriger cette erreur

1. Archivez les journaux existants et supprimez-les de l'ordinateur pour permettre à l'objet **FileLogTraceListener** de créer de nouveaux journaux.
2. Modifiez la valeur de la propriété **MaxFileSize** pour tenir compte des journaux volumineux.
3. Affectez DiscardMessages à la propriété **DiskSpaceExhaustedBehavior** pour ignorer les messages sans avertissement si le journal est trop volumineux.

Voir aussi

Référence

My.Application.Log, objet
My.Log, objet
MaxFileSize
DiskSpaceExhaustedBehavior
FileLogTraceListener

Impossible d'écrire dans le fichier journal, car la quantité d'espace disque libre deviendrait alors inférieure à la valeur ReservedSpace

Unable to write to log file because writing to it would reduce free disk space below ReservedSpace value

La classe FileLogTraceListener n'a pas pu écrire dans le fichier journal pour les raisons suivantes :

- La quantité d'espace disque libre (en octets) est inférieure à la valeur de la propriété ReserveDiskSpace

  — et —

- La valeur de la propriété DiskSpaceExhaustedBehavior est ThrowException.

Pour corriger cette erreur

1. Archivez les journaux existants et supprimez-les de l'ordinateur pour permettre à l'objet **FileLogTraceListener** de créer de nouveaux journaux.
2. Remplacez la valeur de la propriété **ReserveDiskSpace** par une valeur plus petite pour occuper moins d'espace disque.
3. Affectez DiscardMessages à la propriété **DiskSpaceExhaustedBehavior** pour ignorer les messages sans avertissement si l'espace disque libre est insuffisant.

Voir aussi

Référence

My.Application.Log, objet
My.Log, objet
ReserveDiskSpace
DiskSpaceExhaustedBehavior
FileLogTraceListener

Interruption par l'utilisateur

User interrupt occurred

Un utilisateur a appuyé sur CTRL+ATTN ou sur une autre touche d'interruption.

Pour corriger cette erreur

- Recommencez l'opération.

Voir aussi

Concepts

Types d'erreurs

L'utilisation de l'instance par défaut d'une classe dans le constructeur de classe pourrait entraîner des appels récurrents infinis

Use of Default Instance of a class in the class constructor could lead to infinite recursive call

Une instance par défaut d'une classe a été utilisée dans le constructeur de la classe. Cela peut entraîner un appel récurrent infini, qui est également appelé une boucle infinie.

Pour corriger cette erreur

- Supprimez l'instance par défaut du constructeur de la classe.

Voir aussi

Référence

Utilisation de constructeurs et de destructeurs

Utilisez 'FilePutObject' à la place de 'FilePut' quand l'argument est de type 'Object'

Use 'FilePutObject' instead of 'FilePut' when using argument of type 'Object'

La méthode **FilePut** inclut un argument de type **Object**. **FilePutObject** devrait être utilisé à la place de **FilePut** pour éviter toute ambiguïté.

Pour corriger cette erreur

- Remplacez **FilePut** par **FilePutObject**.
- Effectuez un cast de l'argument *Object* en un type plus spécifique.
- Utilisez les fonctionnalités disponibles dans l'objet **My.Computer.FileSystem**.

Voir aussi

Référence

FilePutObject, fonction
My.Computer.FileSystem, objet
My.Computer.FileSystem.WriteAllBytes, méthode

Utilisez 'FileGetObject' à la place de 'FileGet' lorsque l'argument est de type 'Object'

Use 'FileGetObject' instead of 'FileGet' when using argument of type 'Object'

La méthode **FileGet** contient un argument de type **Object**. **FileGetObject** doit être utilisé à la place de **FileGet** pour éviter toute ambiguïté.

Notez que les fonctionnalités fournies par **My.Computer.Filesystem** offrent une facilité d'utilisation et des performances accrues par rapport à **FileGet** ou **FileGetObject**.

Pour corriger cette erreur

1. Remplacez **FileGet** par **FileGetObject**.
2. Effectuez un cast de l'argument *Object* en un type plus spécifique.

Voir aussi

Référence

FileGetObject, fonction
My.Computer.FileSystem, objet

Impossible de combiner VbStrConv.Wide et VbStrConv.Narrow

VbStrConv.Wide and VbStrConv.Narrow cannot be combined

Votre application essaie de combiner les membres d'énumération **VbStrConv Wide** et **Narrow**, qui s'excluent mutuellement.

Pour corriger cette erreur

- Supprimez **VbStrConv.Wide** ou **VbStrConv.Narrow**.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

VbStrConv.Wide et VbStrConv.Narrow ne s'appliquent pas aux paramètres régionaux spécifiés

VbStrConv.Wide and VbStrConv.Narrow are not applicable to the locale specified

L'application essaie d'utiliser les membres d'énumération **VbStrConv Wide** ou **Narrow**, qui ne peuvent pas être appliqués aux paramètres régionaux spécifiés.

Pour corriger cette erreur

- Supprimez **VbStrConv.Wide** ou **VbStrConv.Narrow**.

Voir aussi

Référence

VbStrConv, énumération

System.Globalization Namespace

Concepts

Introduction aux applications internationales basées sur le .NET Framework

Nombre incorrect d'arguments ou assignation de propriété non valide

Wrong number of arguments or property assignment not valid

Une assignation non valide a été tentée.

Pour corriger cette erreur

1. Vérifiez que le nombre d'arguments fourni correspond au nombre requis par la cible.
2. Vérifiez l'assignation de propriété.

Voir aussi

Référence

Opérateurs d'assignation

Vous devez spécifier un nom de fichier

You must specify a file name

Vous avez tenté une opération, telle que le déplacement ou la copie d'un fichier, qui requiert un nom de fichier.

Pour corriger cette erreur

- Entrez le nom de fichier nécessaire.
- Vérifiez que le chemin d'accès ne se termine pas par un caractère de séparation de répertoire (\).

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Référence

My.Computer.FileSystem, objet

Autres ressources

Accès au fichier avec Visual Basic

Vous devez spécifier un nom

You must specify a name

Lorsque vous exécutez des opérations de fichier, telles que l'attribution d'un nouveau nom à un fichier, vous devez spécifier le nom.

Pour corriger cette erreur

- Spécifiez le nom.

Voir aussi

Référence

My.Computer.FileSystem, objet

Concepts

Types d'erreurs

Vous devez spécifier un chemin d'accès se trouvant sous le dossier actif ou un de ses sous-dossiers

You must specify a path that is under the current folder or one of its sub-folders

Le chemin d'accès spécifié n'est pas présent sous le répertoire actif. Cela peut être dû à une analyse incorrecte des chemins d'accès lors de la définition d'un chemin d'accès relatif.

Pour corriger cette erreur

- Assurez-vous que le chemin d'accès correct est fourni.

Voir aussi

Tâches

Comment : analyser des chemins d'accès dans Visual Basic

Outils de ligne de commande de Visual Basic

Cette section décrit le compilateur de ligne de commande et l'outil de mise à niveau de clé Visual Basic.

Dans cette section

Compilateur Visual Basic

Fournit des informations sur le compilateur de ligne de commande qui offre une alternative aux programmes de compilation dans l'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio.

Outil de mise à niveau de mot clé Visual Basic

L'outil de ligne de commande de mise à niveau de clé convertit les fichiers sources Visual Basic qui sont valides en Visual Basic .NET 2002 et 2003 en des fichiers Visual Basic 2005 valides.

Rubriques connexes

Référence Visual Basic

Syntaxe de ligne de commande de Vbupgrade

Compilateur Visual Basic

Le compilateur de ligne de commande Visual Basic fournit une alternative aux programmes de compilation dans l'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio. Cette section décrit les options du compilateur Visual Basic.

Dans cette section

Génération à partir de la ligne de commande (Visual Basic)

Décrit le compilateur de ligne de commande Visual Basic qui est fourni comme une alternative aux programmes de compilation dans l'IDE Visual Studio.

Options du compilateur Visual Basic par ordre alphabétique

Répertorie les options du compilateur par ordre alphabétique dans un tableau.

Options du compilateur Visual Basic par catégorie

Présente les options du compilateur dans des groupes fonctionnels.

Rubriques connexes

Introduction au Concepteur de projets

Explique comment utiliser le Concepteur de projets pour spécifier des paramètres globaux pour votre projet.

Visual Basic

Point de début pour l'aide Visual Basic.

Génération à partir de la ligne de commande (Visual Basic)

Un projet Visual Basic se compose d'un ou plusieurs fichiers sources séparés. Durant le processus de compilation, ces fichiers sont rassemblés sous forme de package (fichier exécutable unique permettant d'exécuter une application).

Le compilateur de ligne de commande Visual Basic fournit une alternative aux programmes de compilation dans l'environnement de développement intégré (IDE, Integrated Development Environment) de Visual Studio. Le compilateur de ligne de commande est prévu pour les situations où vous n'avez pas besoin de l'ensemble des fonctionnalités IDE; par exemple, lorsque vous utilisez ou écrivez du code pour des ordinateurs dont la mémoire système ou l'espace de stockage est limité.

Lorsque la compilation est effectuée à partir de la ligne de commande, vous devez faire référence de manière explicite à la bibliothèque Runtime Microsoft Visual Basic par l'intermédiaire de l'option du compilateur **/reference**.

Pour compiler des fichiers sources à partir de l'IDE de Visual Studio, cliquez dans le menu **Générer** sur la commande **Générer**. Pour compiler des fichiers projet (.vbproj) à la ligne de commande, consultez
Commutateurs de la ligne de commande de Devenv.

Dans cette section

Comment : appeler le compilateur de ligne de commande

Explique comment appeler le compilateur de ligne de commande à l'invite de commande MS-DOS ou à partir d'un sous-répertoire spécifique.

Exemples de lignes de commande de compilation

Fournit une liste d'exemples de lignes de commande que vous pouvez modifier pour vos besoins.

Rubriques connexes

Compilateur Visual Basic

Fournit les listes d'options du compilateur, organisées de manière alphabétique ou selon les besoins.

Vue d'ensemble de la compilation conditionnelle

Explique comment compiler des sections de code précises.

Comment : préparer et gérer des générations

Explique comment organiser le contenu inclus dans les différentes générations, choisir les propriétés de projet et vérifier que les projets sont générés dans l'ordre approprié.

Comment : appeler le compilateur de ligne de commande

Vous pouvez appeler le compilateur de ligne de commande en tapant le nom de son fichier exécutable sur la ligne de commande, également appelée invite de commande MS-DOS. Si vous compilez à partir de l'invite de commande Windows par défaut, vous devez entrer le chemin qualifié complet dans le fichier exécutable. Pour substituer ce comportement par défaut, vous pouvez utiliser l'invite de commande Visual Studio ou modifier la variable d'environnement PATH. Ces deux méthodes vous permettent d'effectuer une compilation à partir d'un répertoire en entrant simplement le nom du compilateur.

Lorsque la compilation est effectuée à partir de la ligne de commande, vous devez faire référence de manière explicite à la bibliothèque Runtime Microsoft Visual Basic par l'intermédiaire de l'option du compilateur **/reference**.

Remarque
Selon vos paramètres actifs ou votre édition, les boîtes de dialogue et les commandes de menu que vous voyez peuvent diffé rer de celles qui sont décrites dans l'aide. Pour modifier vos paramètres, choisissez **Importation et exportation de param ètres** dans le menu **Outils**. Pour plus d'informations, consultez Paramètres Visual Studio.

Pour appeler le compilateur à l'aide de l'invite de commande Visual Studio

1. Ouvrez le dossier de programme Visual Studio Tools dans le groupe de programmes Microsoft Visual Studio.
2. Vous pouvez utiliser l'invite de commande Visual Studio pour accéder au compilateur à partir de n'importe quel répertoire de votre ordinateur, si Visual Studio 2005 est installé.
3. Appelez l'invite de commande Visual Studio.
4. Au niveau de la ligne de commande, tapez le *sourceFileName* de `vbc.exe exe /imports:Microsoft.VisualBasic,System`, puis appuyez sur ENTRÉE.

   Par exemple, si vous avez stocké votre code source dans un répertoire appelé `SourceFiles`, vous devriez ouvrir l'invite de commande et entrer `cd SourceFiles` pour le remplacer par ce répertoire. Si le répertoire contenait un fichier source nommé `Source.vb`, vous pouvez le compiler en tapant `vbc.exe /imports:Microsoft.VisualBasic,System Source.vb`.

Pour affecter le compilateur à la variable d'environnement PATH pour l'invite de commande Windows

1. Utilisez la fonction de recherche de Windows pour rechercher Vbc.exe sur votre disque local.

   Le nom exact du répertoire où se trouve le compilateur dépend de l'emplacement du répertoire Windows et de la version du .NET Compact Framework installée sur l'ordinateur. Si plusieurs versions du .NET Compact Framework sont installées sur votre ordinateur, vous devez déterminer celle qui doit être utilisée (en général, il s'agit de la plus récente).

2. Dans le menu **Démarrer** de votre ordinateur, cliquez avec le bouton droit sur **Poste de travail**, puis cliquez sur **Propriétés** dans le menu contextuel.
3. Cliquez sur l'onglet **Avancé**, puis sur **Variables d'environnement**.
4. Dans le volet **Variables système**, sélectionnez **Chemin d'accès** dans la liste et cliquez sur **Modifier**.
5. Dans la boîte de dialogue **Modifier la variable système**, déplacez le point d'insertion à la fin de la chaîne figurant dans le champ **Valeur de la variable** et tapez un point-virgule (;) suivi du nom complet du répertoire que vous avez trouvé à l'étape 1.
6. Cliquez sur **OK** pour confirmer vos modifications et fermez les boîtes de dialogue.

   Après avoir modifié la variable d'environnement PATH, vous pouvez exécuter le compilateur Visual Basic à l'invite de commande Windows à partir de n'importe quel répertoire de l'ordinateur.

Pour appeler le compilateur à l'aide de l'invite de commande Windows

1. Dans le menu **Démarrer**, cliquez sur le dossier **Accessoires**, puis ouvrez **Invite de commande Windows**.
2. Au niveau de la ligne de commande, tapez le *sourceFileName* de `vbc.exe /imports:Microsoft.VisualBasic,System`, puis appuyez sur ENTRÉE.

Par exemple, si vous avez stocké votre code source dans un répertoire appelé `SourceFiles`, vous devriez ouvrir l'invite de commande et entrer `cd SourceFiles` pour le remplacer par ce répertoire. Si le répertoire contenait un fichier source nommé `Source.vb`, vous pouvez le compiler en tapant `vbc.exe /imports:Microsoft.VisualBasic,System Source.vb`.

Voir aussi

Référence

/reference (Visual Basic)

Concepts

Vue d'ensemble de la compilation conditionnelle

Autres ressources

Compilateur Visual Basic

Exemples de lignes de commande de compilation

Au lieu de compiler les programmes Visual Basic à partir de Visual Studio, vous pouvez les compiler depuis la ligne de commande pour produire des fichiers exécutables (.exe) ou des fichiers de bibliothèques de liens dynamiques (.dll).

Le compilateur de ligne de commande Visual Basic prend en charge un ensemble complet d'options permettant de contrôler les fichiers d'entrée et de sortie, les assemblys, ainsi que les options de débogage et du préprocesseur. Chaque option est disponible sous deux formes interchangeables : **-***option* et **/***option*. Cette documentation ne présente que la forme **/***option*.

Le tableau suivant répertorie certains exemples de lignes de commande que vous pouvez adapter à vos besoins.

Pour	**Utilisez**
Compiler File.vb et créer File.exe	**vbc /reference:Microsoft.VisualBasic.dll File.vb**
Compiler File.vb et créer File.dll	**vbc /target:library File.vb**
Compiler File.vb et créer My.exe	**vbc /out:My.exe File.vb**
Compiler tous les fichiers Visual Basic du répertoire en cours, en activant les optimisations et en définissant le symbole **DEBUG**, afin de produire File2.exe	**vbc /define:DEBUG=1 /optimize /out:File2.exe *.vb**
Compiler tous les fichiers Visual Basic du répertoire en cours, en produisant une version debug de File2.dll sans afficher le logo ou les avertissements	**vbc /target:library /out:File2.dll /nowarn /nologo /debug *.vb**
Compiler tous les fichiers Visual Basic du répertoire en cours en Something.dll	**vbc /target:library /out:Something.dll *.vb**

Lorsque la compilation est effectuée à partir de la ligne de commande, vous devez faire référence de manière explicite à la bibliothèque Runtime Microsoft Visual Basic par l'intermédiaire de l'option du compilateur **/reference**.

Voir aussi

Concepts

Vue d'ensemble de la compilation conditionnelle

Autres ressources

Compilateur Visual Basic

Options du compilateur Visual Basic par ordre alphabétique

Le compilateur de ligne de commande Visual Basic fournit une alternative aux programmes de compilation dans l'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio. La liste suivante répertorie les options du compilateur de ligne de commande Visual Basic par ordre alphabétique.

Option	Objet
@ (Spécifier un fichier réponse)	Spécifie un fichier réponse.
/?	Affiche les options du compilateur. Cette commande est identique à l'option **/help**. Aucune compilation ne se produit.
/addmodule	Entraîne le compilateur à donner des informations de type dans le ou les fichiers spécifiés disponibles pour le projet en cours de compilation.
/baseaddress	Spécifie l'adresse de base d'une DLL.
/bugreport	Crée un fichier qui contient des informations facilitant la rédaction des rapports de bogue.
/codepage	Spécifie la page de codes à utiliser pour tous les fichiers de code source inclus dans la compilation.
/debug	Produit des informations de débogage.
/define	Définit les symboles de compilation conditionnelle.
/delaysign	Spécifie si l'assembly sera complètement ou partiellement signé.
/doc	Traite les commentaires de documentation pour les diriger vers un fichier XML.
/errorreport	Spécifie comment le compilateur de Visual Basic doit rapporter les erreurs internes du compilateur.
/filealign	Spécifie où aligner les sections du fichier de sortie.
/help	Affiche les options du compilateur. Cette commande est identique à l'option **/?**. Aucune compilation ne se produit.
/imports	Importe un espace de noms à partir d'un assembly spécifié.
/keycontainer	Spécifie un nom de conteneur de clé pour une paire de clés afin d'attribuer un nom fort à un assembly.
/keyfile	Spécifie un fichier contenant une clé ou une paire de clés afin d'attribuer un nom fort à un assembly.
/libpath	Spécifie l'emplacement des assemblys référencés par l'option /reference.
/linkresource	Crée un lien vers une ressource managée.
/main	Spécifie la classe qui contient la procédure **Sub Main** à utiliser au démarrage.
/netcf	Paramètre le compilateur afin qu'il cible le .NET Compact Framework.

/noconfig	Ne compilez pas avec Vbc.rsp.
/nologo	Supprime les informations sur la bannière du compilateur.
/nostdlib	Implique que le compilateur ne référence pas les bibliothèques standard.
/nowarn	Désactive la fonction du compilateur permettant de générer des avertissements.
/optimize	Active ou désactive les optimisations du code.
/optioncompare	Détermine si les comparaisons de chaînes doivent être de style binaire ou de type texte spécifique aux paramètres régionaux.
/optionexplicit	Requiert une déclaration explicite des variables.
/optionstrict	Applique une syntaxe de langue stricte.
/out	Spécifie un fichier de sortie.
/platform	Spécifie la plate-forme de processeur que le compilateur cible pour le fichier de sortie.
/quiet	Empêche le compilateur d'afficher du code pour des erreurs et des avertissements liés à la syntaxe.
/recurse	Recherche les fichiers sources à compiler dans les sous-répertoires.
/reference	Importe des métadonnées à partir d'un assembly.
/removeintchecks	Désactive les contrôles de dépassement sur les entiers.
/resource	Incorpore une ressource managée dans un assembly.
/rootnamespace	Spécifie un espace de noms pour toutes les déclarations de type.
/sdkpath	Spécifie l'emplacement de Mscorlib.dll et Microsoft.VisualBasic.dll.
/target	Spécifie le format du fichier de sortie.
/utf8output	Affiche le résultat de la compilation dans le format de codage UTF-8.
/verbose	Sort des informations supplémentaires lors de la compilation.
/warnaserror	Transforme les avertissements en erreurs.
/win32icon	Insère un fichier .ico dans le fichier de sortie.
/win32resource	Insère une ressource Win32 dans le fichier de sortie.

Voir aussi

Référence

Options du compilateur Visual Basic par catégorie

Concepts

Introduction au Concepteur de projets

@ (spécifier un fichier réponse) (Visual Basic)

Spécifie un fichier qui comprend les options du compilateur et les fichiers de code source à compiler.

```
@response_file
```

Arguments
response_file

Requis. Fichier qui répertorie les options du compilateur ou les fichiers de code source à compiler. Mettez le nom de fichier entre guillemets (" ") s'il contient un espace.

Notes

Le compilateur traite les options du compilateur et les fichiers de code source spécifiés dans un fichier réponse de la même façon que s'ils avaient été spécifiés dans la ligne de commande.

Pour spécifier plusieurs fichiers réponse dans une compilation, spécifiez plusieurs options de fichier réponse, telles que dans les exemples suivants.

```
@file1.rsp @file2.rsp
```

Dans un fichier réponse, plusieurs options du compilateur et fichiers de code source peuvent figurer sur une même ligne. En revanche, si vous spécifiez une seule option du compilateur, celle-ci doit obligatoirement figurer sur une seule et même ligne. Les fichiers réponse peuvent contenir des commentaires, précédés du symbole `#`.

Vous pouvez combiner des options spécifiées dans la ligne de commande à des options spécifiées dans un ou plusieurs fichiers réponse. Le compilateur traite les options de commande au fur et à mesure. Par conséquent, des arguments de ligne de commande peuvent substituer des options précédemment affichées dans les fichiers réponse. Inversement, les options d'un fichier réponse substituent les options affichées précédemment dans la ligne de commande ou dans d'autres fichiers réponse.

Visual Basic fournit le fichier Vbc.rsp, qui se trouve dans le même répertoire que le fichier Vbc.exe. Le fichier Vbc.rsp est inclus par défaut, à moins que l'option **/noconfig** soit utilisée. Pour plus d'informations, consultez /noconfig.

Remarque

L'option **@** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Voici quelques lignes extraites d'un exemple de fichier réponse :

```
# build the first output file
/target:exe
/out:MyExe.exe
source1.vb
source2.vb
```

L'exemple suivant montre comment utiliser l'option **@** avec le fichier réponse nommé `File1.rsp`.

```
vbc @file1.rsp
```

Voir aussi
Référence
/noconfig
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/addmodule

Entraîne le compilateur à donner des informations de type dans le ou les fichiers spécifiés disponibles pour le projet en cours de compilation.

```
/addmodule:fileList
```

Arguments
fileList

Requis. Liste de fichiers délimitée par des virgules qui contient des métadonnées, mais pas de manifestes des assemblys. Les noms de fichiers contenant des espaces doivent être mis entre guillemets (" ").

Notes

Les fichiers répertoriés par le paramètre *fileList* doivent être créés à l'aide de l'option **/target:module** ou avec l'équivalent de **/target:module** d'un autre compilateur.

Tous les modules ajoutés par **/addmodule** doivent se trouver dans le même répertoire que le fichier de sortie au moment de l'exécution. Vous pouvez donc spécifier un module dans n'importe quel répertoire au moment de la compilation, mais le module doit se trouver dans le répertoire de l'application au moment de l'exécution. Si tel n'est pas le cas, une erreur TypeLoadException se produit.

Si vous spécifiez (implicitement ou explicitement) une option /target (Visual Basic) différente de **/target:module** avec **/addmodule**, les fichiers passés à **/addmodule** sont inclus dans l'assembly du projet. Un assembly est requis pour exécuter un fichier de sortie contenant un ou plusieurs fichiers ajoutés avec **/addmodule**.

Utilisez /reference (Visual Basic) pour importer les métadonnées à partir d'un fichier qui contient un assembly.

Remarque

L'option **/addmodule** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant crée un module.

VB

```
' t1.vb
' Compile with vbc /target:module t1.vb.
' Outputs t1.netmodule.

Public Class TestClass
    Public i As Integer
End Class
```

Le code suivant importe les types du module.

VB

```
' t2.vb
' Compile with vbc /addmodule:t1.netmodule t2.vb.
Option Strict Off

Namespace NetmoduleTest
    Module Module1
        Sub Main()
            Dim x As TestClass
            x = New TestClass
            x.i = 802
            System.Console.WriteLine(x.i)
        End Sub
```

```
    End Module
End Namespace
```

Lorsque vous exécutez `t1`, la sortie correspond à `802`.

Voir aussi

Référence

/target (Visual Basic)

/reference (Visual Basic)

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/baseaddress

Spécifie une adresse de base par défaut lors de la création d'une DLL.

```
/baseaddress:address
```

Arguments
address

Requis. Adresse de base de la DLL. Cette adresse doit être spécifiée sous la forme d'un nombre hexadécimal.

Notes

L'adresse de base par défaut d'une DLL est définie par le .NET Framework.

Sachez que le dernier chiffre de cette adresse est arrondi. Par exemple, si vous indiquez 0x11110001, ce nombre est arrondi à 0x11110000.

Pour terminer la procédure de signature d'une DLL, utilisez l'option **–R** de l'outil Strong Name (Sn.exe).

Cette option est ignorée si la cible n'est pas une DLL.

Pour définir /baseaddress dans l'IDE de Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Cliquez sur **Avancé**.
4. Modifiez la valeur dans la zone **Adresse de base de la DLL :**.

Remarque
La zone **Adresse de base de la DLL :** est en lecture seule sauf si la cible est une DLL.

Voir aussi
Référence
/target (Visual Basic)
Exemples de lignes de commande de compilation
Outil Strong Name Tool (Sn.exe)
Autres ressources
Compilateur Visual Basic

/bugreport

Crée un fichier qui peut être utilisé lors de la rédaction d'un rapport de bogue.

```
/bugreport:file
```

Arguments
file

Requis. Nom du fichier dans lequel vous souhaitez inclure votre rapport de bogue. Mettez le nom de fichier entre guillemets ("") s'il contient un espace.

Notes

Les informations suivantes sont incluses dans l'argument *file* :

- une copie de tous les fichiers de code source dans la compilation ;
- une liste des options du compilateur utilisée lors de la compilation ;
- des informations sur la version du compilateur, du Common Language Runtime et du système d'exploitation ;
- les résultats de la compilation, le cas échéant ;
- un message vous invitant à décrire le problème ;
- un message vous invitant à décrire la façon dont le problème pourrait être résolu, selon vous.

Dans la mesure où une copie de tous les fichiers de code source est placée dans *file*, vous pouvez reproduire l'erreur (supposée) dans le code avec le programme le plus court possible.

Remarque de sécurité

L'option **/bugreport** produit un fichier qui contient des informations potentiellement sensibles comme l'heure actuelle, la version du compilateur, la version du .NET Framework, la version du système d'exploitation, le nom d'utilisateur, les arguments de la ligne de commande ayant permis l'exécution du compilateur, l'ensemble du code source et la forme binaire de tout assembly référencé. Cette option est accessible en spécifiant des options de ligne de commande dans le fichier Web.config pour une compilation côté serveur d'une application ASP.NET. Pour empêcher cela, modifiez le fichier Machine.config afin d'interdire aux utilisateurs de faire une compilation sur le serveur.

Si cette option est utilisée avec **/errorreport:prompt**, **/errorreport:queue** ou **/errorreport:send** et si votre application rencontre une erreur interne du compilateur, les informations dans *file* sont alors envoyées à Microsoft Corporation. Ces informations aideront les ingénieurs Microsoft à identifier la cause de l'erreur et pourront peut-être permettre d'améliorer la version finale suivante de Visual Basic. Par défaut, aucune information n'est envoyée à Microsoft. Toutefois, lorsque vous compilez une application avec **/errorreport:queue**, qui est activé par défaut, l'application rassemble ses rapports d'erreurs. Ensuite, lorsque l'administrateur de l'ordinateur se connecte, le système de rapport d'erreurs affiche une fenêtre indépendante qui autorise l'administrateur à envoyer à Microsoft tous les rapports d'erreurs créés depuis la connexion.

Remarque

L'option **/bugreport** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

L'exemple suivant compile `T2.vb` et place toutes les informations de rapport de bogue dans le fichier `Problem.txt`.

```
vbc /bugreport:problem.txt t2.vb
```

Voir aussi
Référence
/debug (Visual Basic)

/errorreport
Exemples de lignes de commande de compilation
trustLevel, élément de securityPolicy (Schéma des paramètres ASP.NET)
Autres ressources
Compilateur Visual Basic

/codepage (Visual Basic)

Spécifie la page de codes à utiliser pour tous les fichiers de code source inclus dans la compilation.

```
/codepage:id
```

Arguments
id

Requis. Le compilateur utilise la page de codes spécifiée par *id* pour interpréter le codage des fichiers sources.

Notes

Pour compiler code source enregistré avec un codage spécifique, vous pouvez utiliser **/codepage** pour spécifier quelle page de codes doit être utilisée. L'option **/codepage** s'applique à tous les fichiers de code source inclus dans votre compilation. Pour plus d'informations, consultez Prise en charge du codage pour les pages de codes.

L'option **/codepage** n'est pas nécessaire si les fichiers de code de source ont été enregistrés à l'aide de la page de codes ANSI actuelle, Unicode ou UTF-8 avec une signature. Visual Studio enregistre par défaut tous les fichiers de code de source avec la page de codes ANSI actuelle, à moins que l'utilisateur spécifie un autre codage dans la boîte de dialogue **Codage**. Visual Studio utilise la boîte de dialogue **Codage** pour ouvrir des fichiers de code de source enregistrés avec une page de codes différente. Pour plus d'informations, consultez Codage, boîte de dialogue.

Remarque

L'option **/codepage** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Voir aussi
Référence
Codage, boîte de dialogue
Autres ressources
Compilateur Visual Basic

/debug (Visual Basic)

Entraîne le compilateur à générer des informations de débogage et à les placer dans le ou les fichiers de sortie.

```
/debug[+ | -]
' -or-
/debug:[full | pdbonly]
```

Arguments

+ | **-**

Facultatif. Si vous spécifiez **+** ou **/debug**, le compilateur générera des informations de débogage et les placera dans un fichier .pdb. Spécifier **-** équivaut à ne pas spécifier **/debug**.

full | **pdbonly**

Facultatif. Spécifie le type d'informations de débogage générées par le compilateur. Si vous ne spécifiez pas **/debug:pdbonly**, la valeur par défaut est **full**, ce qui vous permet d'attacher un débogueur au programme en cours d'exécution. L'argument **pdbonly** permet un débogage du code source lorsque le programme est démarré dans le débogueur, mais affiche du code en langage assembleur uniquement lorsque le programme en cours d'exécution est attaché au débogueur.

Notes

Utilisez cette option pour créer des générations de débogage. Si vous ne spécifiez pas **/debug**, **/debug+** ou **/debug:full**, vous ne pourrez pas déboguer le fichier de sortie de votre programme.

Par défaut, les informations de débogage ne sont pas émises (**/debug-**). Pour émettre des informations de débogage, spécifiez **/debug** ou **/debug+**.

Pour plus d'informations sur la configuration des performances de débogage d'une application, consultez Simplification du débogage d'une image.

Pour définir /debug dans l'environnement de développement intégré Visual Studio

1. Un projet étant sélectionné dans l'**Explorateur de solutions**, cliquez dans le menu **Projet** sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Cliquez sur **Avancé**.
4. Modifiez la valeur dans la zone **Adresse de base de la DLL**.

Exemple

L'exemple suivant place des informations de débogage dans le fichier de sortie `App.exe`.

```
vbc /debug /out:app.exe test.vb
```

Voir aussi

Référence

/bugreport

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/define (Visual Basic)

Définit des constantes de compilation conditionnelle.

```
/define:["]symbol=value[,symbol=value]["]
' -or-
/d:["]symbol=value[,symbol=value]["]
```

Arguments
symbol

Requis. Le symbole à définir.

value

Requis. La valeur assignée à l'argument *symbol*. Si *value* est une chaîne, elle doit être entourée par des séquences de barre oblique inverse/guillemets (\") plutôt que par des guillemets.

Notes

L'option **/define** a un effet semblable à l'utilisation d'une directive de préprocesseur **#Const** dans votre fichier source, excepté que les constantes définies avec **/define** sont publiques et s'appliquent à tous les fichiers dans le projet.

Vous pouvez utiliser les symboles créés par cette option avec la directive **#If Then #Else** pour compiler des fichiers sources de manière conditionnelle.

/d est la forme abrégée de **/define**.

Vous pouvez définir plusieurs symboles avec l'option **/define** en séparant les définitions de symbole par des virgules.

Pour définir /define dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Cliquez sur **Avancé**.
4. Modifiez la valeur dans la zone **Constantes personnalisées**.

Exemple

Le code suivant définit puis utilise deux constantes de compilation conditionnelle :

VB

```
    ' Vbc /define:DEBUGMODE=True,TRAPERRORS=False test.vb
    Sub mysub()
#If debugmode Then
        ' Insert debug statements here.
         MsgBox("debug mode")
#Else
        ' Insert default statements here.
#End If
    End Sub
```

Voir aussi
Référence
#If...Then...#Else, directives
#Const, directive
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/delaysign

Spécifie si l'assembly sera complètement ou partiellement signé.

```
/delaysign[+ | -]
```

Argument
+ | **-**

Facultatif. Utilisez **/delaysign-** si vous souhaitez obtenir un assembly complètement signé. Utilisez **/delaysign+** si vous souhaitez placer la clé publique dans l'assembly et réservez de l'espace pour le hachage signé. La valeur par défaut est **/delaysign-**.

Notes

L'option **/delaysign** est sans effet sauf si elle est utilisée avec /keyfile ou /keycontainer.

Lorsque vous demandez un assembly complètement signé, le compilateur hache le fichier contenant le manifeste (métadonnées de l'assembly) et signe ce hachage avec la clé privée. La signature numérique obtenue est stockée dans le fichier qui contient le manifeste. Lorsque la signature d'un assembly est différée, le compilateur ne calcule ni ne stocke la signature, mais réserve un espace dans le fichier pour pouvoir y ajouter ultérieurement la signature.

Par exemple, à l'aide de **/delaysign+**, le développeur d'une organisation peut distribuer des versions de test d'un assembly non signées que les testeurs peuvent enregistrer dans le Global Assembly Cache et utiliser. Lorsqu'elle a fini de travailler sur l'assembly, la personne responsable de la clé privée de l'organisation peut le signer complètement. Ce compartimentage protège la clé privée de l'organisation contre la divulgation tout en permettant à tous les développeurs de travailler sur les assemblys.

Pour plus d'informations sur la signature d'un assembly, consultez Création et utilisation d'assemblys avec nom fort.

Pour définir /temporiser la signature dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Signature**.
3. Définissez la valeur dans la zone **Temporiser la signature uniquement**.

Voir aussi
Référence
/keyfile
/keycontainer
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/doc

Traite les commentaires de documentation pour les diriger vers un fichier XML.

```
/doc[+ | -]
' -or-
/doc:file
```

Arguments

+ | **-**

Facultatif. Si vous spécifiez +, ou simplement **/doc**, le compilateur générera des informations de documentation et les placera dans un fichier XML. La spécification de **-** revient à ne pas spécifier **/doc** ; ainsi, aucune information de documentation n'est créée.

file

Requis si **/doc:** est utilisé. Spécifie le fichier de sortie XML, qui est rempli avec les commentaires extraits des fichiers de code source de la compilation. Si le nom de fichier contient un espace, placez-le entre guillemets (" ").

Notes

L'option **/doc** contrôle si le compilateur génère un fichier XML contenant des commentaires de documentation. Si vous utilisez la syntaxe **/doc:***file*, le paramètre *file* spécifie le nom du fichier XML. Si vous utilisez **/doc** ou **/doc+**, le compilateur tire le nom du fichier XML à partir du fichier exécutable ou de la bibliothèque que le compilateur crée. Si vous utilisez **/doc-** ou que vous ne spécifiez pas l'option **/doc**, le compilateur ne crée pas de fichier XML.

Dans les fichiers de code source, les commentaires de documentation peuvent précéder les définitions suivantes :

- Types définis par l'utilisateur, tels qu'une classe ou une interface
- Des membres, tels qu'un champ, un événement, une propriété, une fonction ou une sous-routine.

Pour utiliser le fichier XML généré avec la fonctionnalité IntelliSense de Visual Studio, autorisez le nom du fichier XML à être identique à celui de l'assembly que vous souhaitez prendre en charge. Assurez-vous que le fichier XML est dans le même répertoire que l'assembly afin que le fichier .xml soit également recherché lorsque l'assembly est référencé dans le projet Visual Studio. Les fichiers de documentation XML ne sont pas requis pour qu'IntelliSense fonctionne pour le code dans un projet ou dans des projets référencés par un projet.

À moins que vous compiliez avec **/target:module**, le fichier XML contient les balises `<assembly></assembly>`. Ces balises spécifient le nom du fichier contenant le manifeste d'assembly pour le fichier de sortie de la compilation.

Consultez Balises XML recommandées pour les commentaires de documentation (Visual Basic) pour savoir comment générer la documentation à partir de commentaires dans votre code.

Pour définir /doc dans l'environnement de développement intégré de Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Définissez la valeur dans la zone **Générer le fichier de documentation XML**.

Exemple

Consultez Documentation de votre code avec le langage XML (Visual Basic) pour obtenir un exemple.

Voir aussi

Concepts

Documentation de votre code avec le langage XML (Visual Basic)

Autres ressources

Compilateur Visual Basic

/errorreport

Spécifie comment le compilateur Visual Basic doit rapporter les erreurs internes du compilateur.

```
/errorreport:{ prompt | queue | send | none }
```

Notes

Cette option offre un moyen pratique de signaler une erreur interne du compilateur Visual Basic à l'équipe Visual Basic de Microsoft. Par défaut, le compilateur n'envoie aucune information à Microsoft. Toutefois, si vous rencontrez une erreur interne du compilateur, cette option vous permet de la signaler à Microsoft. Ces informations aideront les ingénieurs Microsoft à identifier la cause et pourront peut-être permettre d'améliorer la version finale suivante de Visual Basic.

La capacité d'un utilisateur à envoyer des rapports dépend de l'ordinateur et des autorisations de la stratégie de l'utilisateur.

Le tableau suivant résume l'effet de l'option **/errorreport**.

Option	**Comportement**
prompt	Si une erreur interne du compilateur se produit, une boîte de dialogue apparaît afin que vous puissiez consulter les données exactes collectées par le compilateur. Vous pouvez déterminer si des informations sensibles sont contenues dans le rapport d'erreurs et prendre une décision quant à son envoi à Microsoft. Si vous décidez d'envoyer le rapport et si les paramètres de stratégie de l'ordinateur et de l'utilisateur l'autorisent, le compilateur envoie les données à Microsoft.
queue	Met en file d'attente le rapport d'erreurs. Lorsque vous vous connectez avec des droits d'administrateur, une fenêtre indépendante s'affiche pour vous permettre de signaler toute défaillance depuis votre dernière connexion (vous ne serez pas invité plus d'une fois tous les trois jours à envoyer des rapports pour les défaillances). C'est le comportement par défaut lorsque vous ne spécifiez pas l'option **/errorreport**.
send	Si une erreur interne du compilateur se produit et si les paramètres de stratégie de l'ordinateur et de l'utilisateur l'autorisent, le compilateur envoie les données à Microsoft.
none	Si une erreur interne du compilateur se produit, elle n'est pas collectée ou envoyée à Microsoft.

Les données envoyées par le compilateur contiennent la pile au moment de l'erreur, qui renferme d'habitude du code source. Si **/errorreport** est utilisé avec l'option /bugreport, le fichier source entier est transmis.

Il est préférable d'utiliser cette option avec l'option /bugreport, car elle permet aux ingénieurs Microsoft de reproduire l'erreur plus facilement.

Remarque
L'option **/errorreport** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant essaie de compiler `T2.vb`, et si le compilateur rencontre une erreur interne du compilateur, il vous invite à

envoyer le rapport d'erreurs à Microsoft.

```
vbc /errorreport:prompt t2.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

/bugreport

Autres ressources

Compilateur Visual Basic

/filealign

Spécifie l'emplacement dans lequel aligner les sections du fichier de sortie.

```
/filealign:number
```

Arguments
number

Requis. Valeur spécifiant l'alignement des sections du fichier de sortie. Les valeurs valides sont 512, 1024, 2048, 4096 et 8192. Ces valeurs sont exprimées en octets.

Notes

Vous pouvez utiliser l'option **/filealign** pour spécifier l'alignement de sections dans votre fichier de sortie. Les sections représentent des blocs de mémoire contiguë dans un fichier exécutable portable (PE, Portable Executable) qui contient du code ou des données. L'option **/filealign** vous permet de compiler votre application avec un alignement non standard ; la plupart des développeurs n'ont pas besoin de cette option.

Chaque section est alignée sur une limite qui est un multiple de la valeur **/filealign**. Il n'y a pas de valeur par défaut fixe. Si **/filealign** n'est pas spécifié, le compilateur sélectionne la valeur par défaut au moment de la compilation.

En spécifiant la taille de la section, vous pouvez modifier la taille du fichier de sortie. Il est parfois utile de modifier la taille de section pour les programmes appelés à s'exécuter sur des périphériques plus petits.

Remarque

L'option **/filealign** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Voir aussi
Autres ressources
Compilateur Visual Basic

/help, /? (Visual Basic)

Affiche les options du compilateur.

```
/help
' -or-
/?
```

Notes

Si cette option est incluse dans une compilation, aucun fichier de sortie n'est créé et aucune compilation ne se produit.

Remarque

L'option **/help** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant affiche l'aide à partir de la ligne de commande.

```
vbc /help
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/imports (Visual Basic)

Importe un espace de noms à partir d'un assembly spécifié.

```
/imports:namespaceList
```

Arguments
namespaceList

Requis. Liste d'espaces de noms délimitée par des virgules à importer.

Notes

L'option **/imports** permet d'importer n'importe quel espace de noms défini dans l'ensemble de fichiers sources en cours ou à partir de tout assembly référencé.

Les membres d'un espace de noms spécifiés avec **/imports** sont disponibles pour tous les fichiers de code source de la compilation. L'instruction Imports vous permet d'utiliser un espace de noms dans un seul fichier de code source.

Pour définir /imports dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Référence**.
3. Entrez le nom de l'espace de noms dans la zone située à côté du bouton **Ajouter une importation utilisateur**.
4. Cliquez sur le bouton **Ajouter une importation utilisateur**.

Exemple

Le code suivant compile si **/imports:system** est spécifié :

VB

```
Module MyModule
    Sub Main()
        Console.WriteLine("test")
        ' Otherwise, would need
        ' System.Console.WriteLine("test")
    End Sub
End Module
```

Voir aussi
Référence
Exemples de lignes de commande de compilation
Concepts
Références et l'instruction Imports
Autres ressources
Compilateur Visual Basic

/keycontainer

Spécifie un nom de conteneur de clé pour une paire de clés afin d'attribuer un nom fort à un assembly.

```
/keycontainer:container
```

Arguments
container

Requis. Fichier conteneur qui contient la clé. Mettez le nom de fichier entre guillemets ("") s'il contient un espace.

Notes

Le compilateur crée le composant qui peut être partagé en insérant une clé publique dans le manifeste d'assembly et en signant l'assembly final avec la clé privée. Pour générer un fichier de clé, tapez `sn -k` *file* sur la ligne de commande. L'option `-i` installe la paire de clés dans un conteneur. Pour plus d'informations, consultez Outil Strong Name Tool (Sn.exe).

Si vous compilez à l'aide de **/target:module**, le nom du fichier de clé est conservé dans le module et incorporé dans l'assembly qui est créé lorsque vous compilez un assembly à l'aide de /addmodule.

Vous pouvez également spécifier cette option comme attribut personnalisé (AssemblyKeyNameAttribute) dans le code source de n'importe quel module MSIL (Microsoft Intermediate Language).

Vous pouvez également passer vos informations de chiffrement au compilateur avec /keyfile. Utilisez /delaysign si vous souhaitez obtenir un assembly partiellement signé.

Pour plus d'informations sur la signature d'un assembly, consultez Création et utilisation d'assemblys avec nom fort.

Remarque

L'option **/keycontainer** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile le fichier source `Input.vb` et spécifie un conteneur de clés.

```
vbc /keycontainer:key1 input.vb
```

Voir aussi
Référence
/keyfile
Exemples de lignes de commande de compilation
Concepts
Assemblys
Autres ressources
Compilateur Visual Basic

/keyfile

Spécifie un fichier contenant une clé ou une paire de clés afin d'attribuer un nom fort à un assembly.

```
/keyfile:file
```

Arguments
file

Requis. Le fichier qui comprend la clé. Si le nom de fichier contient un espace, mettez le nom entre guillemets (" ").

Notes

Le compilateur insère la clé publique dans le manifeste de l'assembly et signe ensuite l'assembly final avec la clé privée. Pour générer un fichier de clé, tapez `sn -k file` sur la ligne de commande. Pour plus d'informations, consultez Outil Strong Name Tool (Sn.exe).

Si vous compilez à l'aide de **/target:module**, le nom du fichier de clé est conservé dans le module et incorporé dans l'assembly qui est créé lorsque vous compilez un assembly à l'aide de /addmodule.

Vous pouvez également passer vos informations de chiffrement au compilateur avec /keycontainer. Utilisez /delaysign si vous souhaitez obtenir un assembly partiellement signé.

Vous pouvez également spécifier cette option comme attribut personnalisé (AssemblyKeyFileAttribute) dans le code source de n'importe quel module Microsoft Intermediate Language.

Si **/keyfile** et /keycontainer sont tous deux spécifiés (par une option de ligne de commande ou par un attribut personnalisé) dans la même compilation, le compilateur essaie d'abord le conteneur de clé. Si cette tentative réussit, l'assembly est signé avec les informations figurant dans le conteneur de clé. Si le compilateur ne trouve pas le conteneur de clé, il essaie le fichier spécifié avec **/keyfile**. Si cette tentative réussit, l'assembly est signé avec les informations du fichier de clé et les informations de clé sont installées dans le conteneur de clé (résultat similaire à celui de **sn -i**) afin que, lors de la prochaine compilation, le conteneur de clé soit valide.

Notez qu'un fichier de clé pourrait contenir uniquement la clé publique.

Pour plus d'informations sur la signature d'un assembly, consultez Création et utilisation d'assemblys avec nom fort.

Remarque

L'option **/keyfile** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile le fichier source `Input.vb` et spécifie un fichier de clé.

```
vbc /keyfile:myfile.sn input.vb
```

Voir aussi
Référence
/reference (Visual Basic)
Exemples de lignes de commande de compilation
Concepts
Assemblys
Autres ressources
Compilateur Visual Basic

/libpath

Spécifie l'emplacement des assemblys référencés.

```
/libpath:dirList
```

Arguments
dirList

Requis. Liste délimitée par des points-virgules des répertoires où le compilateur doit vérifier si un assembly référencé ne se trouve pas dans le répertoire de travail actif (répertoire depuis lequel vous appelez le compilateur) ou dans le répertoire système du Common Language Runtime. Si le nom de répertoire contient un espace, placez-le entre guillemets (" ").

Notes

L'option **/libpath** spécifie l'emplacement des assemblys référencés par l'option /reference.

Le compilateur recherche les références d'assembly qui ne sont pas qualifiées complètes dans l'ordre suivant :

1. Répertoire de travail en cours. Il s'agit du répertoire à partir duquel le compilateur est appelé.
2. Répertoire système du Common Language Runtime.
3. Répertoires spécifiés par **/libpath**.
4. Répertoires spécifiés par la variable d'environnement LIB.

L'option **/libpath** peut être cumulée ; si vous la spécifiez plus d'une fois, elle sera ajoutée aux valeurs précédentes.

Utilisez l'option **/reference** pour spécifier une référence d'assembly.

Pour définir /libpath dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Références**.
3. Cliquez sur le bouton **Chemins d'accès des références...**.
4. Dans la boîte de dialogue **Chemins d'accès des références**, entrez le nom du répertoire dans la zone **Dossier :**.
5. Cliquez sur **Ajouter un dossier**.

Exemple

Le code suivant compile `T2.vb` pour créer un fichier .exe. Le compilateur recherche les références d'assembly dans le répertoire de travail, dans le répertoire racine du lecteur C: et dans le répertoire New Assemblies du lecteur C:.

```
vbc /libpath:c:\;"c:\New Assemblies" /reference:t2.dll t2.vb
```

Voir aussi
Référence
Exemples de lignes de commande de compilation
Concepts
Assemblys
Autres ressources
Compilateur Visual Basic

/linkresource (Visual Basic)

Crée un lien vers une ressource managée.

```
/linkresource:filename[,identifier[,public|private]]
' -or-
/linkres:filename[,identifier[,public|private]]
```

Arguments
filename

Requis. Fichier de ressources à lier à l'assembly. Si le nom de fichier contient un espace, mettez le nom entre guillemets (" ").

identifier

Facultatif. Nom logique de la ressource. Nom utilisé pour charger la ressource. La valeur par défaut est le nom du fichier. Si vous le souhaitez, vous pouvez spécifier si le fichier est public ou privé dans le manifeste d'assembly, par exemple :`/linkres:filename.res,myname.res,public`. Par défaut, *filename* est public dans l'assembly.

Notes

L'option **/linkresource** n'incorpore pas le fichier de ressources dans le fichier de sortie ; utilisez plutôt l'option **/resource** pour cela.

L'option **/linkresource** nécessite une option **/target** différente de **/target:module**.

Si *filename* est un fichier de ressources .NET Framework créé, par exemple, par le Outil Resource File Generator Tool (Resgen.exe) ou dans l'environnement de développement, il est accessible à l'aide de membres de l'espace de noms System.Resources. (Pour plus d'informations, consultez ResourceManager.) Pour accéder à toutes les autres ressources au moment de l'exécution, utilisez les méthodes qui commencent par **GetManifestResource** dans la classe Assembly.

Le nom de fichier peut être de n'importe quel format de fichier. Par exemple, vous pouvez décider d'intégrer une DLL native dans l'assembly : ainsi, elle pourra être installée dans le Global Assembly Cache et sera accessible depuis le code managé de l'assembly.

La forme abrégée de **/linkresource** est **/linkres**.

Remarque

L'option **/linkresource** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `In.vb` et le lie au fichier de ressources `Rf.resource` :

```
vbc /linkresource:rf.resource in.vb
```

Voir aussi
Référence
/target (Visual Basic)
/resource (Visual Basic)
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/main

Spécifie la classe ou le module qui contient la procédure **Sub Main**.

```
/main:location
```

Arguments
location

Requis. Qualification complète de la classe ou du module contenant la procédure **Sub Main** à appeler au démarrage du programme. Celle-ci peut être sous la forme **/main:module** ou **/main:namespace.module**.

Notes

Utilisez cette option lorsque vous créez un fichier exécutable ou un programme exécutable Windows. Si vous omettez l'option **/main**, le compilateur recherche une procédure **Sub Main** partagée valide dans tous les modules et toutes les classes publiques.

Consultez Procédure Main dans Visual Basic pour obtenir des explications sur les différentes formes de la procédure **Main**.

Lorsque *location* est une classe qui hérite de Form, le compilateur fournit une procédure **Main** par défaut qui démarre l'application si la classe n'a aucune procédure **Main**. Cela vous permet de compiler le code à partir de la ligne de commande créée dans l'environnement de développement.

VB

```
' Compile with /r:System.dll,SYSTEM.WINDOWS.FORMS.DLL /main:MyC
Public Class MyC
    Inherits System.Windows.Forms.Form
End Class
```

Pour définir /main dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**.

   Pour plus d'informations, consultez Introduction au Concepteur de projets.

2. Cliquez sur l'onglet **Application**.

3. Vérifiez que la case à cocher **Activer l'infrastructure de l'application** n'est pas activée.

4. Modifiez la valeur dans la zone **Objet de démarrage**.

Exemple

Le code suivant compile `T2.vb` et `T3.vb`, en spécifiant que la procédure **Sub Main** se trouve dans la classe `Test2` :

```
vbc t2.vb t3.vb /main:Test2
```

Voir aussi
Référence
/target (Visual Basic)
Exemples de lignes de commande de compilation
Concepts
Version Visual Basic de Hello World!
Procédure Main dans Visual Basic
Autres ressources
Compilateur Visual Basic

/netcf

Définit le compilateur afin qu'il cible le .NET Compact Framework.

```
/netcf
```

Notes

L'option **/netcf** provoque que le compilateur Visual Basic afin qu'il cible le .NET Compact Framework plutôt que le .NET Framework complet. La fonctionnalité de langue qui est seulement présente sur le .NET Framework complet est désactivée.

L'option **/netcf** est conçue pour être utilisée avec /sdkpath. Les fonctionnalités de langue désactivées par **/netcf** sont les mêmes que celles qui sont absentes des fichiers ciblés avec **/sdkpath**.

Remarque

L'option **/netcf** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement l ors de la compilation à partir de la ligne de commande. L'option **/netcf** est définie lorsqu'un projet smart device Visual Basic est chargé.

L'option **/netcf** modifie les fonctionnalités de langue suivantes :

- Le mot clé End (Visual Basic), qui met fin à l'exécution d'un programme, est désactivé. Le programme suivant compile et exécute sans **/netcf**, mais échoue au moment de la compilation avec **/netcf**.

  VB

  ```
  Module Module1
      Sub Main()
          End   ' not valid to terminate execution with /netcf
      End Sub
  End Module
  ```

- La liaison tardive, sous quelle forme que ce soit, est désactivée. Les erreurs de compilation sont générées lorsque des scénarios de liaison tardive sont rencontrés. Le programme suivant compile et exécute sans **/netcf**, mais échoue au moment de la compilation avec **/netcf**.

  VB

  ```
  Class LateBoundClass
      Sub S1()
      End Sub

      Default Property P1(ByVal s As String) As Integer
          Get
          End Get
          Set(ByVal Value As Integer)
          End Set
      End Property
  End Class

  Module Module1
      Sub Main()
          Dim o1 As Object
          Dim o2 As Object
          Dim o3 As Object
          Dim IntArr(3) As Integer
  ```

```
        o1 = New LateBoundClass
        o2 = 1
        o3 = IntArr

        ' Late-bound calls
        o1.S1()
        o1.P1("member") = 1

        ' Dictionary member access
        o1!member = 1

        ' Late-bound overload resolution
        LateBoundSub(o2)

        ' Late-bound array
        o3(1) = 1
    End Sub

    Sub LateBoundSub(ByVal n As Integer)
    End Sub

    Sub LateBoundSub(ByVal s As String)
    End Sub
End Module
```

- Les modificateurs Auto, Ansi et Unicode (Visual Basic) sont désactivés. La syntaxe de l'instruction Declare est également modifiée comme suit : `Declare Sub|Function name Lib "library" [Alias "alias"] [([arglist])]`. Le code suivant montre l'effet qu'exerce **/netcf** sur une compilation.

  VB

```
' compile with: /target:library
Module Module1
    ' valid with or without /netcf
    Declare Sub DllSub Lib "SomeLib.dll" ()

    ' not valid with /netcf
    Declare Auto Sub DllSub1 Lib "SomeLib.dll" ()
    Declare Ansi Sub DllSub2 Lib "SomeLib.dll" ()
    Declare Unicode Sub DllSub3 Lib "SomeLib.dll" ()
End Module
```

- L'utilisation des mots clés Visual Basic 6.0 qui ont été supprimés de Visual Basic génère une erreur différente lorsque **/netcf** est utilisé. Cela affecte les messages d'erreur qui apparaissent pour les mots clés suivants :
  - **Open**
  - **Close**
  - **Put**
  - **Print**
  - **Write**
  - **Input**
  - **Lock**
  - **Unlock**

- **Seek**
- **Width**
- **Name**
- **FreeFile**
- **EOF**
- **Loc**
- **LOF**
- **Line**

Exemple

Le code suivant compile `Myfile.vb` avec le .NET Compact Framework à l'aide des versions de Mscorlib.dll et Microsoft.VisualBasic.dll trouvées dans le répertoire d'installation par défaut du .NET Compact Framework situé sur le lecteur C. En général, il est préférable d'utiliser la version la plus récente du .NET Compact Framework.

```
vbc /netcf /sdkpath:"c:\Program Files\Microsoft Visual Studio .NET 2003\CompactFrameworkSDK
\v1.0.5000\Windows CE " myfile.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

/sdkpath

Autres ressources

Compilateur Visual Basic

/noconfig

Spécifie que le compilateur ne doit pas référencer automatiquement les assemblys du .NET Framework couramment utilisés ou importer les espaces de noms **System** et **Microsoft.VisualBasic**.

```
/noconfig
```

Notes

L'option **/noconfig** indique au compilateur de ne pas compiler à l'aide du fichier Vbc.rsp qui se situe dans le même répertoire que le fichier Vbc.exe. Le fichier Vbc.rsp référence les assemblys du .NET Framework communément utilisés et importe les espaces de noms **System** et **Microsoft.VisualBasic**. Le compilateur référence implicitement l'assembly System.dll à moins que l'option **/nostdlib** soit spécifiée. L'option **/nostdlib** indique au compilateur de ne pas compiler à l'aide du fichier Vbc.rsp ou de référencer automatiquement l'assembly System.dll.

Remarque
Les assemblys Mscorlib.dll et Microsoft.VisualBasic.dll sont toujours référencés.

Vous pouvez modifier le fichier Vbc.rsp pour spécifier des options du compilateur supplémentaires qui doivent être incluses dans chaque compilation Vbc.exe (sauf en cas de spécification de l'option **/noconfig**). Pour plus d'informations, consultez @ (spécifier un fichier réponse) (Visual Basic).

Le compilateur traite les dernières options passées à la commande **vbc**. En conséquence, toute option incluse sur la ligne de commande substitue la définition de la même option dans le fichier Vbc.rsp.

Remarque
L'option **/noconfig** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Voir aussi

Référence

/nostdlib (Visual Basic)

@ (spécifier un fichier réponse) (Visual Basic)

/reference (Visual Basic)

Autres ressources

Compilateur Visual Basic

/nologo (Visual Basic)

Supprime l'affichage de la page de copyright et des messages d'informations lors de la compilation.

```
/nologo
```

Notes

Si vous spécifiez **/nologo**, le compilateur n'affiche pas la page de copyright. Par défaut, l'option **/nologo** n'est pas activée.

Remarque

L'option **/nologo** n'est pas accessible dans l'environnement de développement Visual Studio 2005. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `T2.vb` et n'affiche pas de bannière de copyright.

```
vbc /nologo t2.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/nostdlib (Visual Basic)

Empêche le compilateur de référencer automatiquement les bibliothèques standard.

```
/nostdlib
```

Notes

L'option **/nostdlib** supprime la référence automatique à l'assembly System.dll et empêche le compilateur de lire le fichier Vbc.rsp. Le fichier Vbc.rsp, qui se trouve dans le même répertoire que le fichier Vbc.exe, référence les assemblys du .NET Framework communément utilisés et importe les espaces de noms **System** et **Microsoft.VisualBasic**.

Remarque

Les assemblys Mscorlib.dll et Microsoft.VisualBasic.dll sont toujours référencés.

Remarque

L'option **/nostdlib** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `T2.vb` sans référencer les bibliothèques standard. Vous devez définir la constante de compilation conditionnelle **_MYTYPE** à la valeur de chaîne "Empty" pour supprimer l'objet **My**.

```
vbc /nostdlib /define:_MYTYPE=\"Empty\" T2.vb
```

Voir aussi

Référence

/noconfig

Exemples de lignes de commande de compilation

Concepts

Personnalisation de la disponibilité ou non des objets dans My

Autres ressources

Compilateur Visual Basic

/nowarn

Désactive la fonction du compilateur permettant de générer des avertissements.

```
/nowarn[:numberList]
```

Arguments
numberList

Facultatif. Liste délimitée par des virgules des numéros d'ID d'avertissement que le compilateur doit supprimer. Si les ID d'avertissement ne sont pas spécifiés, tous les avertissements sont supprimés.

Notes

L'option **/nowarn** a pour conséquence que le compilateur ne génère pas d'avertissement. Pour supprimer un avertissement, fournissez l'ID d'avertissement à l'option **/nowarn** qui suit les deux-points. Séparez les numéros des avertissements par une virgule.

Il suffit de spécifier la partie numérique de l'identificateur de l'avertissement. Par exemple, si vous souhaitez supprimer BC42024, numéro d'avertissement pour les variables locales inutilisées, spécifiez `/nowarn:42024`.

Pour plus d'informations sur les numéros d'ID d'avertissement, consultez Configuration d'avertissements en Visual Basic.

Pour définir /nowarn dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Activez la case à cocher **Désactiver tous les avertissements** pour désactiver tous les avertissements.

   - ou -

   Pour désactiver un avertissement particulier, cliquez sur **Aucun** dans la liste déroulante adjacente à l'avertissement.

Exemple

Le code suivant compile `T2.vb` et n'affiche pas d'avertissements.

```
vbc /nowarn t2.vb
```

Le code suivant compile `T2.vb` et n'affiche pas d'avertissements pour les variables locales inutilisées (42024).

```
vbc /nowarn:42024 t2.vb
```

Voir aussi
Référence
Exemples de lignes de commande de compilation
Concepts
Configuration d'avertissements en Visual Basic
Autres ressources
Compilateur Visual Basic

/optimize

Active ou désactive les optimisations du compilateur.

```
/optimize[ + | - ]
```

Arguments

+ | -

Facultatif. L'option **/optimize-** désactive les optimisations du compilateur. L'option **/optimize+** active les optimisations. Par défaut, les optimisations sont désactivées.

Notes

Les optimisations du compilateur diminuent la taille du fichier de sortie, le rendent plus rapide et plus efficace. Toutefois, les optimisations entraînant une réorganisation du code dans le fichier de sortie, la valeur **/optimize+** peut rendre le débogage difficile.

Tous les modules générés à l'aide **/target:module** pour un assembly doivent utiliser les mêmes paramètres **/optimize** que l'assembly. Pour plus d'informations, consultez /target (Visual Basic).

Vous pouvez combiner les options **/optimize** et **/debug**.

Pour définir /optimize dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**.

   Pour plus d'informations, consultez Introduction au Concepteur de projets.

2. Cliquez sur l'onglet **Compiler**.

3. Cliquez sur le bouton **Avancé**.

4. Modifiez la case à cocher **Activer les optimisations**.

Exemple

Le code suivant compile `T2.vb` et active les optimisations du compilateur.

```
vbc t2.vb /optimize
```

Voir aussi

Référence

/debug (Visual Basic)
Exemples de lignes de commande de compilation
/target (Visual Basic)

Autres ressources

Compilateur Visual Basic

/optioncompare

Spécifie la façon dont sont effectuées les comparaisons de chaînes.

```
/optioncompare:{binary | text}
```

Notes

Vous pouvez spécifier **/optioncompare** sous l'une des deux formes suivantes : **/optioncompare:binary** pour utiliser des comparaisons de chaînes binaires et **/optioncompare:text** pour utiliser des comparaisons de chaînes de texte. Par défaut, le compilateur utilise **/optioncompare:binary**.

Sous Microsoft Windows, la page de code en cours d'utilisation détermine le tri binaire. Un ordre de tri binaire par défaut est :

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

Les comparaisons de chaîne de texte sont basées selon un tri ne respectant pas la casse du texte, déterminé par les paramètres régionaux de votre système. Un ordre de tri de texte par défaut est :

```
(A = a) < (À = à) < (B=b) < (E=e) < (Ê = ê) < (Z=z) < (Ø = ø)
```

Pour définir /optioncompare dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Modifiez la valeur dans la zone **Option Compare**.

Pour définir /optioncompare par programme

- Consultez la rubrique Option Compare, instruction.

Exemple

Le code suivant compile `ProjFile.vb` et utilise des comparaisons de chaînes binaires :

```
vbc /optioncompare:binary projFile.vb
```

Voir aussi

Référence

/optionexplicit
/optionstrict
Exemples de lignes de commande de compilation
Option Compare, instruction
Valeurs par défaut VB, Projets, boîte de dialogue Options

Autres ressources

Compilateur Visual Basic

/optionexplicit

Entraîne le compilateur à signaler des erreurs si les variables ne sont pas déclarées préalablement à leur utilisation.

```
/optionexplicit[+ | -]
```

Arguments

+ | -

Facultatif. Spécifiez **/optionexplicit+** pour requérir une déclaration explicite des variables. L'option **/optionexplicit+** est la valeur par défaut et il s'agit de la même que **/optionexplicit**. L'option **/optionexplicit-** permet une déclaration implicite des variables.

Notes

Pour définir /optionexplicit dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Modifiez la valeur dans la zone **Option Explicit**.

Pour définir /optionexplicit par programme

- Consultez Option Explicit, instruction (Visual Basic).

Exemple

Le code suivant compile si **/optionexplicit-** est utilisé.

VB

```
Module Module1
    Sub Main()
        i = 99
        System.Console.WriteLine(i)
    End Sub
End Module
```

Voir aussi

Référence

/optioncompare
/optionstrict
Exemples de lignes de commande de compilation
Option Explicit, instruction (Visual Basic)
Valeurs par défaut VB, Projets, boîte de dialogue Options

Autres ressources

Compilateur Visual Basic

/optionstrict

Applique une sémantique de type stricte pour limiter les conversions de type implicites.

```
/optionstrict[+ | -]
/optionstrict[:custom]
```

Arguments

+ | -

Facultatif. L'option **/optionstrict+** restreint les conversions de type implicites. La valeur par défaut pour cette option est **/optionstrict-**. L'option **/optionstrict+** est identique à **/optionstrict**. Vous pouvez utiliser les deux pour la sémantique de type permissive.

custom

Requis. Avertir lorsque la syntaxe de langue stricte n'est pas respectée.

Notes

Lorsque **/optionstrict+** est appliqué, seules les conversions étendues peuvent être effectuées implicitement. Les conversions restrictives implicites, telles que l'assignation d'un objet de type **Decimal** à un objet de type entier, sont signalées comme des erreurs.

Pour générer des avertissements pour les conversions restrictives implicites, utilisez **/optionstrict:custom**. Utilisez **/nowarn:numberlist** pour ignorer certains avertissements et **/warnaserror:numberlist** pour traiter d'autres comme des erreurs.

Pour définir /optionstrict dans l'IDE de Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Modifiez la valeur dans la zone **Option Strict**.

Pour définir /optionstrict par programme

- Consultez Option Strict, instruction.

Exemple

Le code suivant compile `Test.vb` en utilisant une sémantique de type stricte.

```
vbc /optionstrict+ test.vb
```

Voir aussi

Référence

/optioncompare
/optionexplicit
/nowarn
/warnaserror (Visual Basic)
Exemples de lignes de commande de compilation
Option Strict, instruction
Valeurs par défaut VB, Projets, boîte de dialogue Options

Autres ressources

Compilateur Visual Basic

/out (Visual Basic)

Spécifie le nom du fichier de sortie.

```
/out:filename
```

Arguments
filename

Requis. Nom du fichier de sortie créé par le compilateur. Si le nom de fichier contient un espace, mettez le nom entre guillemets (" ").

Notes

Le compilateur est censé trouver le fichier de sortie qui suit l'option **/out**.

Spécifiez le nom complet et l'extension du fichier à créer. Dans le cas contraire, le fichier .exe tire son nom du fichier de code source contenant la procédure **Sub Main**, et le fichier .dll tire son nom du premier fichier de code source.

Pour définir /out dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Application**.
3. Modifiez la valeur dans la zone **Nom de l'assembly**.

Exemple

Le code suivant compile `T2.vb` et crée un fichier de sortie `T2.exe`.

```
vbc t2.vb /out:t3.exe
```

Voir aussi
Référence
/target (Visual Basic)
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/platform (Visual Basic)

Spécifie quelle version de plate-forme du Common Language Runtime (CLR) peut exécuter le fichier de sortie.

```
/platform:{ x86 | x64 | Itanium | anycpu }
```

Argument

anycpu

Compile votre assembly pour qu'il s'exécute sur toutes les plates-formes. C'est le comportement par défaut lorsque l'option **/platform** n'est pas spécifiée.

x86

Compile votre assembly pour qu'il soit exécuté par le CLR compatible 32 bits, x86.

x64

Compile votre assembly pour qu'il soit exécuté par le CLR 64 bits sur un ordinateur qui prend en charge le jeu d'instructions AMD64 ou EM64T.

Itanium

Compile votre assembly pour qu'il soit exécuté par le CLR 64 bits sur un ordinateur doté du processeur Itanium.

Notes

Utilisez l'option **/platform** pour spécifier le type de processeur ciblé par le fichier de sortie.

En règle générale, les assemblys .NET Framework écrits en Visual Basic s'exécuteront de la même façon quelle que soit la plate-forme. Toutefois, dans certains cas, les comportements sont différents sur des plates-formes différentes. Ces cas sont les suivants :

- Des structures contenant des membres qui changent de taille selon la plate-forme (par exemple tous les types de pointeur).
- Des opérations arithmétiques sur les pointeurs qui incluent des tailles de constantes.
- Des appels de plate-forme ou des déclarations COM incorrects qui utilisent **Integer** pour les handles au lieu de IntPtr.
- Cast de **IntPtr** en **Integer**.
- Utilisation de l'appel de plate-forme ou de COM Interop avec les composants qui n'existent pas sur toutes les plates-formes.

L'option **/platform** atténuera certains problèmes si vous savez que vous avez fait des hypothèses à propos de l'architecture sur laquelle le code s'exécutera. Plus précisément :

- Si vous décidez de cibler une plate-forme 64 bits et que l'application est exécutée sur un ordinateur 32 bits, le message d'erreur s'affiche plus tôt et est plus axé sur le problème que l'erreur qui se produirait sans utilisation de ce commutateur.
- Si vous définissez l'option 32 bits sur le commutateur et que l'application est exécutée par la suite sur un ordinateur 64 bits, elle s'exécutera dans le sous-système WOW au lieu de s'exécuter en mode natif.

Sur un système d'exploitation Windows 64 bits :

- les assemblys compilés avec **/platform:x86** s'exécutent sur le CLR 32 bits qui s'exécute sous WOW64 ;
- les fichiers exécutables compilés avec **/platform:anycpu** s'exécutent sur le CLR 64 bits ;
- une DLL compilée avec **/platform:anycpu** s'exécute sur le même CLR que le processus dans lequel elle est chargée.

Pour plus d'informations sur le développement d'une application à exécuter sur un système d'exploitation Windows 64 bits, consultez Applications 64 bits.

Pour définir /nowarn dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Cliquez sur le bouton **Avancé**.
4. Modifiez la valeur de la zone de liste déroulante **Unité centrale cible**.

Exemple

L'exemple suivant indique comment utiliser **/platform** pour spécifier que l'application ne doit être exécutée que par le CLR 64 bits sur un système d'exploitation Windows 64 bits pour Itanium.

```
vbc /platform:Itanium myItanium.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/quiet

Empêche le compilateur d'afficher du code pour des erreurs et des avertissements liés à la syntaxe.

```
/quiet
```

Notes

Par défaut, **/quiet** n'est pas activée. Lorsque le compilateur signale une erreur ou un avertissement lié à la syntaxe, il renvoie également la ligne du code source. Pour les applications qui analysent les résultats de la compilation, il est parfois plus pratique que le compilateur exporte uniquement le texte du diagnostic.

Dans l'exemple suivant, `Module1` extrait une erreur qui inclut le code source compilé sans **/quiet**.

```
Module Module1
    Sub Main()
        x()
    End Sub
End Module
```

Sortie :

```
E:\test\t2.vb(3) : error BC30451: Name 'x' is not declared.

x

~
```

Lorsque ce code est compilé avec **/quiet**, le compilateur extrait uniquement ce qui suit :

```
E:\test\t2.vb(3) : error BC30451: Name 'x' is not declared.
```

Remarque

L'option **/quiet** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement l ors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `T2.vb` et n'affiche pas de code pour les diagnostics du compilateur liés à la syntaxe :

```
vbc /quiet t2.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/recurse

Compile des fichiers de code source dans tous les répertoires enfants du répertoire spécifié ou du répertoire du projet.

```
/recurse:[dir\]file
```

Arguments

dir

Facultatif. Répertoire dans lequel vous voulez commencer la recherche. Si ce répertoire n'est pas spécifié, la recherche commence dans le répertoire du projet.

file

Requis. Le ou les fichiers à rechercher. Les caractères génériques sont autorisés.

Notes

Vous pouvez utiliser des caractères génériques dans un nom de fichier pour compiler tous les fichiers correspondants dans le répertoire du projet sans utiliser **/recurse**. Si aucun nom de fichier de sortie n'est spécifié, le compilateur base le nom du fichier de sortie sur le premier fichier d'entrée traité. Généralement, c'est le premier fichier de la liste de fichiers compilés lorsqu'ils sont affichés par ordre alphabétique. Par conséquent, il est préférable de spécifier un fichier de sortie à l'aide de l'option **/out**.

Remarque

L'option **/recurse** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile tous les fichiers Visual Basic dans le répertoire en cours.

```
vbc *.vb
```

Le code suivant compile tous les fichiers Visual Basic dans le répertoire `Test\ABC` et dans tous les répertoires en dessous de celui-ci, puis génère `Test.ABC.dll`.

```
vbc /target:library /out:Test.ABC.dll /recurse:Test\ABC\*.vb
```

Voir aussi

Référence

/out (Visual Basic)

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/reference (Visual Basic)

Entraîne le compilateur à donner des informations de type dans les assemblys spécifiés disponibles pour le projet en cours de compilation.

```
/reference:fileList
' -or-
/r:fileList
```

Arguments
fileList

Requis. Liste délimitée par des virgules de noms de fichiers d'assembly. Si le nom de fichier contient un espace, mettez le nom entre guillemets.

Notes

Le ou les fichiers que vous importez doivent comprendre des métadonnées de l'assembly. Seuls les types publics sont visibles à l'extérieur de l'assembly. L'option /addmodule importe des métadonnées à partir d'un module.

En cas de référence à un assembly (Assembly A) faisant lui-même référence à un autre assembly (Assembly B), vous devez référencer Assembly B si :

- Un type utilisé à partir de l'assembly A hérite d'un type ou implémente une interface à partir de l'assembly B.
- Un champ, une propriété, un événement ou une méthode dont le type de retour ou de paramètre est issu de l'assembly B est appelé.

Utilisez l'option /libpath pour spécifier le répertoire dans lequel se trouvent une ou plusieurs de vos références d'assembly.

Pour que le compilateur reconnaisse un type dans un assembly (pas dans un module), il doit être forcé de résoudre le type. Pour cela, vous pouvez, par exemple, définir une instance du type. D'autres méthodes permettent au compilateur de résoudre les noms de types dans un assembly. Par exemple, si vous héritez d'un type dans un assembly, le compilateur reconnaît ensuite le nom de type.

Le fichier réponse Vbc.rsp, qui référence les assemblys .NET Framework couramment utilisés, est utilisé par défaut. Utilisez **/noconfig** si vous ne voulez pas que le compilateur utilise Vbc.rsp.

La forme abrégée de **/reference** est **/r**.

Pour définir /reference dans l'environnement de développement intégré Visual Studio

- Consultez la rubrique Ajouter une référence, boîte de dialogue.

Exemple

Le code suivant compile le fichier source `Input.vb` et fait référence à des assemblys issus de `Metad1.dll` et `Metad2.dll` pour produire `Out.exe` :

```
vbc /reference:metad1.dll,metad2.dll /out:out.exe input.vb
```

Voir aussi
Référence
/noconfig
/target (Visual Basic)
Public (Visual Basic)
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/removeintchecks

Active ou désactive la vérification d'erreurs de dépassement durant les opérations sur les entiers.

```
/removeintchecks[+ | -]
```

Arguments

+ | **-**

Facultatif. L'option **/removeintchecks-** entraîne le compilateur à vérifier tous les calculs d'entier pour rechercher des erreurs telles que le dépassement de capacité ou la division par zéro. La valeur par défaut est **/removeintchecks-**.

Si vous spécifiez **/removeintchecks** ou **/removeintchecks+**, la vérification d'erreurs ne sera pas activée et les calculs d'entier pourront être plus rapides. Cependant, sans une vérification des erreurs, des résultats erronés peuvent être stockés sans être signalés si les capacités de type données sont dépassées.

Pour définir /removeintchecks dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Cliquez sur le bouton **Avancé**.
4. Modifiez la valeur de la zone **Supprimer les contrôles de dépassement sur les entiers**.

Exemple

Le code suivant compile `Test.vb` et désactive la vérification des erreurs de dépassement sur les entiers.

```
vbc /removeintchecks+ test.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/resource (Visual Basic)

Incorpore une ressource managée dans un assembly.

```
/resource:filename[,identifier[,public|private]]
' -or-
/res:filename[,identifier[,public|private]]
```

Arguments
filename

Requis. Nom du fichier de ressources à incorporer dans le fichier de sortie. Par défaut, *filename* est public dans l'assembly. Mettez le nom de fichier entre guillemets (" ") s'il contient un espace.

identifier

Facultatif. Nom logique de la ressource, c'est-à-dire le nom utilisé pour charger cette dernière. La valeur par défaut est le nom du fichier. Vous pouvez éventuellement spécifier si la ressource est publique ou privée dans le manifeste de l'assembly ; par exemple : **/res:***filename.res*,*myname.res*,**public**

Notes

Utilisez **/linkresource** pour lier une ressource à un assembly sans placer le fichier de ressources dans le fichier de sortie.

Si *filename* est un fichier de ressources .NET Framework créé, par exemple, par le Outil Resource File Generator Tool (Resgen.exe) ou dans l'environnement de développement, il est accessible avec des membres de l'espace de noms System.Resources (pour plus d'informations, consultez ResourceManager). Pour accéder à toutes les autres ressources au moment de l'exécution, utilisez l'une des méthodes suivantes : GetManifestResourceInfo, GetManifestResourceNames ou GetManifestResourceStream.

La forme abrégée de **/resource** est **/res**.

Pour définir /resource dans l'environnement de développement intégré Visual Studio

- Consultez Ressources, page du Concepteur de projets.

Exemple

Le code suivant compile `In.vb` et le joint au fichier de ressources `Rf.resource`.

```
vbc /res:rf.resource in.vb
```

Voir aussi
Référence
/win32resource
/linkresource (Visual Basic)
/target (Visual Basic)
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/rootnamespace

Spécifie un espace de noms pour toutes les déclarations de type.

```
/rootnamespace:namespace
```

Arguments
namespace

Nom de l'espace de noms dans lequel vous voulez inclure toutes les déclarations de type pour le projet en cours.

Notes

Si vous utilisez le fichier exécutable Visual Studio (Devenv.exe) pour compiler un projet créé dans l'environnement de développement intégré Visual Studio, utilisez **/rootnamespace** pour spécifier la valeur de la propriété RootNamespace. Pour plus d'informations, consultez Commutateurs de la ligne de commande de Devenv.

Utilisez le désassembleur MSIL du common language runtime (`Ildasm.exe`) pour afficher les noms d'espaces de noms dans votre fichier de sortie.

Pour définir/référencer dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Application**.
3. Modifiez la valeur dans la zone **Espace de noms racine**.

Exemple

Le code suivant compile `In.vb` et place toutes les déclarations de type dans l'espace de noms `mynamespace`.

```
vbc /rootnamespace:mynamespace in.vb
```

Voir aussi
Référence
MSIL Disassembler (Ildasm.exe)
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/sdkpath

Spécifie l'emplacement de Mscorlib.dll et Microsoft.VisualBasic.dll.

```
/sdkpath:path
```

Arguments
path

Répertoire contenant les versions de Mscorlib.dll et de Microsoft.VisualBasic.dll à utiliser pour la compilation. Ce chemin d'accès n'est pas vérifié tant qu'il n'est pas chargé. Mettez le nom de répertoire entre guillemets (" ") s'il contient un espace.

Notes

Cette option demande au compilateur Visual Basic de charger les fichiers Mscorlib.dll et Microsoft.VisualBasic.dll à partir d'un emplacement autre que celui utilisé par défaut. L'option **/sdkpath** a été conçue pour être utilisée avec /netcf. Le .NET Compact Framework utilise différentes versions de ces bibliothèques d'assistance pour éviter l'utilisation de types et de fonctionnalités de langue introuvables sur les périphériques.

Remarque

L'option **/sdkpath** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande. L'option **/sdkpath** est définie lorsqu'un projet smart device Visual Basic est chargé.

Exemple

Le code suivant compile `Myfile.vb` avec le .NET Compact Framework à l'aide des versions de Mscorlib.dll et Microsoft.VisualBasic.dll trouvées dans le répertoire d'installation par défaut du .NET Compact Framework situé sur le lecteur C. En général, il est préférable d'utiliser la version la plus récente du .NET Compact Framework.

```
vbc /netcf /sdkpath:"c:\Program Files\Microsoft Visual Studio .NET 2003\CompactFrameworkSDK
\v1.0.5000\Windows CE " myfile.vb
```

Voir aussi
Référence
Exemples de lignes de commande de compilation
/netcf
Autres ressources
Compilateur Visual Basic

/target (Visual Basic)

Spécifie le format de sortie du compilateur.

```
/target:{exe | library | module | winexe}
```

Notes

Le tableau suivant résume l'effet de l'option **/target**.

Option	Comportement
/target:exe	Entraîne le compilateur à créer une application console exécutable. C'est l'option par défaut lorsque vous ne spécifiez pas l'option **/target**. Le fichier exécutable est créé avec l'extension .exe. Sauf si vous spécifiez un autre nom avec l'option **/out**, le fichier de sortie prend le nom du fichier d'entrée contenant la procédure **Sub Main**. Seule une procédure **Sub Main** est requise dans les fichiers du code source compilés dans un fichier .exe. Utilisez l'option du compilateur **/main** pour spécifier quelle classe contient la procédure **Sub Main**.
/target:library	Provoque la création d'une bibliothèque de liens dynamiques (DLL) par le compilateur. Le fichier de bibliothèque de liens dynamiques est créé avec l'extension .dll. Sauf si vous spécifiez l'option **/out**, le nom du fichier de sortie prend le nom du premier fichier d'entrée. Une procédure **Sub Main** n'est pas nécessaire lors de la génération d'une DLL.
/target:module	Entraîne le compilateur à générer un module qui peut être ajouté à un assembly. Le fichier de sortie est créé avec une extension .netmodule. Le Common Language Runtime .NET ne peut pas charger un fichier qui ne contient pas d'assembly. Cependant, un tel fichier peut être incorporé dans le manifeste d'assembly d'un assembly en utilisant l'option **/reference**. Lorsque le code d'un module référence des types internes dans un autre module, les deux modules doivent être incorporés dans un manifeste d'assembly en utilisant **/reference**. L'option /addmodule importe des métadonnées à partir d'un module.
/target:winexe	Entraîne le compilateur à créer un programme exécutable Windows. Le fichier exécutable est créé avec l'extension .exe. Un programme Windows est un programme qui fournit une interface utilisateur à partir de la bibliothèque de classe .NET Framework ou avec les API Win32. Sauf si vous spécifiez un autre nom avec l'option **/out**, le fichier de sortie prend le nom du fichier d'entrée contenant la procédure **Sub Main**. Seule une procédure **Sub Main** est requise dans les fichiers du code source compilés dans un fichier .exe. Dans les cas où votre code possède plusieurs classes avec une procédure **Sub Main**, utilisez l'option du compilateur **/main** pour spécifier quelle classe contient la procédure **Sub Main**

Avec l'option **/target**, un manifeste d'assembly .NET Framework est placé dans un fichier de sortie, sauf si vous spécifiez **/target:module**.

Chaque instance de Vbc.exe produit un fichier de sortie au plus. Si vous spécifiez une option du compilateur (par exemple, **/out** ou **/target**) plusieurs fois, c'est la dernière option traitée par le compilateur qui est appliquée. Le manifeste reçoit les informations sur tous les fichiers d'une compilation. Tous les fichiers de sortie sauf ceux créés avec **/target:module** peuvent contenir des métadonnées de l'assembly dans le manifeste. Utilisez MSIL Disassembler (Ildasm.exe) pour visualiser les métadonnées dans un fichier de sortie.

La forme abrégée de **/target** est **/t**.

Pour définir/cibler dans l'environnement de développement intégré Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Application**.
3. Modifiez la valeur dans la zone **Type d'application**.

Remarque
L'option **/target:module** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `in.vb`, en créant `in.dll` :

```
vbc /target:library in.vb
```

Voir aussi

Référence

/main
/out (Visual Basic)
/reference (Visual Basic)
/addmodule
Exemples de lignes de commande de compilation

Concepts

Assemblys

Autres ressources

Compilateur Visual Basic

/utf8output (Visual Basic)

Affiche le résultat de la compilation dans le format de codage UTF-8.

```
/utf8output[+ | -]
```

Arguments

+ | **-**

Facultatif. La valeur par défaut de cette option est **/utf8output-**, ce qui signifie que les résultats de la compilation n'utilisent pas le codage UTF-8. Les options **/utf8output** et **/utf8output+** sont équivalentes.

Notes

Dans certaines configurations internationales, les résultats de la compilation ne peuvent pas être affichés correctement dans la console. Le cas échéant, utilisez **/utf8output** et redirigez la sortie du compilateur vers un fichier.

Remarque

L'option **/utf8output** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `In.vb` et affiche la sortie du compilateur dans le format de codage UTF-8.

```
vbc /utf8output in.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/verbose

Entraîne le compilateur à produire un état détaillé et des messages d'erreur.

```
/verbose[+ | -]
```

Arguments

+ | **-**

Facultatif. Spécifier **/verbose** équivaut à spécifier **/verbose+**, ce qui entraîne le compilateur à émettre des messages documentés. La valeur par défaut pour cette option est **/verbose-**.

Notes

L'option **/verbose** affiche des informations sur le nombre total d'erreurs émises par le compilateur, signale les assemblys chargés par un module et affiche les fichiers en cours de compilation.

Remarque

L'option **/verbose** n'est pas accessible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `In.vb` et demande au compilateur d'afficher des informations d'état détaillées :

```
vbc /verbose in.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Autres ressources

Compilateur Visual Basic

/warnaserror (Visual Basic)

Entraîne le compilateur à traiter la première occurrence d'un avertissement comme une erreur.

```
/warnaserror[+ | -][:numberList]
```

Arguments

+ | -

Facultatif. Par défaut, **/warnaserror-** est en vigueur, les avertissements n'empêchent pas le compilateur de générer un fichier de sortie. Si vous utilisez l'option **/warnaserror** , qui est identique à **/warnaserror+**, les avertissements sont considérés comme des erreurs.

numberList

Facultatif. Liste délimitée par des virgules des numéros d'ID d'avertissement auxquels l'option **/warnaserror** s'applique. Si aucun ID d'avertissement n'est spécifié, l'option **/warnaserror** s'applique à tous les avertissements.

Notes

Avec l'option **/warnaserror**, tous les avertissements sont considérés comme des erreurs. Ainsi, les messages qui normalement devraient être affichés en tant qu'avertissements apparaissent comme des erreurs. Le compilateur signale les occurrences ultérieures du même avertissement en tant qu'avertissements.

Par défaut, **/warnaserror-** est en vigueur, ce qui engendre des avertissements de type informatif uniquement. Si vous utilisez l'option **/warnaserror** , qui est identique à **/warnaserror+**, les avertissements sont considérés comme des erreurs.

Si vous souhaitez que seuls certains avertissements spécifiques soient considérés comme des erreurs, vous pouvez spécifier une liste séparée par des virgules répertoriant les numéros d'avertissement à traiter comme des erreurs.

Remarque

L'option **/warnaserror** ne contrôle pas l'affichage des avertissements. Utilisez l'option /nowarn pour désactiver des avertissements.

Pour définir /warnaserror pour que tous les avertissements soient traités comme des erreurs dans l'IDE de Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Compiler**.
3. Assurez-vous que la case à cocher **Désactiver tous les avertissements** est désactivée.
4. Activez la case à cocher **Considérer tous les avertissements comme des erreurs**.

Pour définir /warnaserror pour que les avertissements spécifiques soient traités comme des erreurs dans l'IDE de Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**.
2. Cliquez sur l'onglet **Compiler**.
3. Assurez-vous que la case à cocher **Désactiver tous les avertissements** est désactivée.
4. Assurez-vous que la case à cocher **Considérer tous les avertissements comme des erreurs** est désactivée.
5. Sélectionnez **Erreur** dans la colonne **Notification** adjacente à l'avertissement qui doit être traité comme une erreur.

Exemple

Le code suivant compile `In.vb` et demande au compilateur d'afficher une erreur pour la première occurrence de chaque avertissement rencontré :

```
vbc /warnaserror in.vb
```

Le code suivant compile `T2.vb` et les traite uniquement l'avertissement pour les variables locales inutilisées (42024) comme une erreur.

```
vbc /warnaserror:42024 t2.vb
```

Voir aussi

Référence

Exemples de lignes de commande de compilation

Concepts

Configuration d'avertissements en Visual Basic

Autres ressources

Compilateur Visual Basic

/win32icon

Insère un fichier .ico dans le fichier de sortie. Ce fichier .ico représente le fichier de sortie dans l'**Explorateur Windows**.

```
/win32icon:filename
```

Arguments
filename

Fichier .ico à ajouter au fichier de sortie. Mettez le nom de fichier entre guillemets (" ") s'il contient un espace.

Notes

Vous pouvez créer un fichier .ico avec le compilateur de ressources (RC) Microsoft Windows. Le compilateur de ressources est appelé lors de la compilation d'un programme Visual C++ ; un fichier .ico est alors créé à partir du fichier .rc. Les options **/win32icon** et **/win32resource** s'excluent mutuellement.

Consultez /linkresource (Visual Basic) pour référencer un fichier de ressources .NET Framework ou /resource (Visual Basic) pour joindre un fichier de ressources .NET Framework. Consultez /win32resource pour importer un fichier .res.

Pour définir /win32icon dans l'IDE de Visual Studio

1. Sélectionnez un projet dans l'**Explorateur de solutions**. Dans le menu **Projet**, cliquez sur **Propriétés**. Pour plus d'informations, consultez Introduction au Concepteur de projets.
2. Cliquez sur l'onglet **Application**.
3. Modifiez la valeur dans la zone **Icône**.

Exemple

Le code suivant compile `In.vb` et attache un fichier .ico `Rf.ico`.

```
vbc /win32icon:rf.ico in.vb
```

Voir aussi
Référence
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

/win32resource

Insère un fichier de ressources Win32 dans le fichier de sortie.

```
/win32resource:filename
```

Arguments
filename

Nom du fichier de ressources à ajouter à votre fichier de sortie. Mettez le nom de fichier entre guillemets (" ") s'il contient un espace.

Notes

Vous pouvez créer un fichier de ressources Win32 avec le compilateur de ressources (RC) Microsoft Windows.

Une ressource Win32 peut contenir des informations de version ou des bitmaps (icônes) qui permettent d'identifier facilement votre application dans l'**Explorateur Windows**. Si vous ne spécifiez pas **/win32resource**, le compilateur génère des informations sur la version en fonction de la version de l'assembly. Les options **/win32resource** et **/win32icon** s'excluent mutuellement.

Consultez /linkresource (Visual Basic) pour référencer un fichier de ressources .NET Framework ou /resource (Visual Basic) pour joindre un fichier de ressources .NET Framework.

Remarque

L'option **/win32resource** n'est pas disponible dans l'environnement de développement Visual Studio. Elle est disponible uniquement lors de la compilation à partir de la ligne de commande.

Exemple

Le code suivant compile `In.vb` et le lie au fichier de ressources Win32 `Rf.res` :

```
vbc /win32resource:rf.res in.vb
```

Voir aussi
Référence
Exemples de lignes de commande de compilation
Autres ressources
Compilateur Visual Basic

Options du compilateur Visual Basic par catégorie

Le compilateur de ligne de commande Visual Basic fournit une alternative aux programmes de compilation dans l'environnement de développement intégré (IDE, Integrated Development Environment) Visual Studio. La liste suivante répertorie les options du compilateur de ligne de commande Visual Basic triées par catégorie fonctionnelle.

Résultats de la compilation

Option	Objet
/nologo	Supprime les informations sur la bannière du compilateur.
/utf8output	Affiche le résultat de la compilation dans le format de codage UTF-8.
/verbose	Sort des informations supplémentaires lors de la compilation.

Optimisation

Option	Objet
/filealign	Spécifie où aligner les sections du fichier de sortie.
/optimize	Active ou désactive les optimisations.

Fichiers de sortie

Option	Objet
/doc	Traiter les commentaires de documentation pour les diriger vers un fichier XML.
/netcf	Définit le compilateur afin qu'il cible le .NET Compact Framework.
/out	Spécifie un fichier de sortie.
/target	Spécifie le format de la sortie.

Assemblys .NET

Option	Objet
/addmodule	Entraîne le compilateur à donner des informations de type dans le ou les fichiers spécifiés disponibles pour le projet en cours de compilation.
/delaysign	Spécifie si l'assembly sera complètement ou partiellement signé.
/imports	Importe un espace de noms à partir d'un assembly spécifié.
/keycontainer	Spécifie un nom de conteneur de clé pour une paire de clés afin d'attribuer un nom fort à un assembly.
/keyfile	Spécifie un fichier contenant une clé ou une paire de clés afin d'attribuer un nom fort à un assembly.
/libpath	Spécifie l'emplacement des assemblys référencés par l'option /reference.
/reference	Importe des métadonnées à partir d'un assembly.

Débogage/Vérification des erreurs

Option	Objet
/bugreport	Crée un fichier qui contient des informations facilitant la rédaction des rapports de bogue.
/debug	Produit des informations de débogage.

/nowarn	Désactive la fonction du compilateur permettant de générer des avertissements.
/quiet	Empêche le compilateur d'afficher du code pour des erreurs et des avertissements liés à la syntaxe.
/removeintchecks	Désactive les contrôles de dépassement sur les entiers.
/warnaserror	Transforme les avertissements en erreurs.

Aide

Option	Objet
/?	Affiche les options du compilateur. Cette commande est identique à l'option **/help**. Aucune compilation ne se produit.
/help	Affiche les options du compilateur. Cette commande est identique à l'option **/?**. Aucune compilation ne se produit.

Langage

Option	Objet
/optionexplicit	Requiert une déclaration explicite des variables.
/optionstrict	Applique une sémantique de type stricte.
/optioncompare	Détermine si les comparaisons de chaînes doivent être de style binaire ou de type texte spécifique aux param ètres régionaux.

Préprocesseur

Option	Objet
/define	Définit les symboles de compilation conditionnelle.

Ressources

Option	Objet
/linkresource	Crée un lien vers une ressource managée.
/resource	Incorpore une ressource managée dans un assembly.
/win32icon	Insère un fichier .ico dans le fichier de sortie.
/win32resource	Insère une ressource Win32 dans le fichier de sortie.

Divers

Option	Objet
@ (spécifier un fichier réponse)	Spécifie un fichier réponse.
/baseaddress	Spécifie l'adresse de base d'une DLL.
/codepage	Spécifie la page de codes à utiliser pour tous les fichiers de code source inclus dans la compilat ion.
/errorreport	Spécifie comment le compilateur Visual Basic doit rapporter les erreurs internes du compilateu r.
/main	Spécifie la classe qui contient la procédure **Sub Main** à utiliser au démarrage.
/noconfig	Ne compilez pas avec Vbc.rsp
/nostdlib	Implique que le compilateur ne référence pas les bibliothèques standard.

/platform	Spécifie la plate-forme de processeur que le compilateur cible pour le fichier de sortie.
/recurse	Recherche les fichiers sources à compiler dans les sous-répertoires.
/rootnamespace	Spécifie un espace de noms pour toutes les déclarations de type.
/sdkpath	Spécifie l'emplacement de Mscorlib.dll et Microsoft.VisualBasic.dll.

Voir aussi

Référence

Options du compilateur Visual Basic par ordre alphabétique

Concepts

Introduction au Concepteur de projets

Outil de mise à niveau de mot clé Visual Basic

L'outil de ligne de commande Mise à niveau de mot clé rend les fichiers sources Visual Basic .NET incompatibles avec la version actuelle de Visual Basic.

```
vb7to8[.exe] [arguments]
```

Arguments

L'ordre d'apparition des arguments suivants n'a pas d'importance.

Argument	Signification
filename	Provoque le traitement du fichier *filename* par l'outil. Mettez le nom de fichier entre guillemets doubles (".") s'il contient un espace. L'outil peut traiter plusieurs fichiers.
/check	Indique les fichiers qui doivent être mis à niveau sans les modifier. Si aucune mise à niveau n'est nécessaire, l'outil retourne « Aucun fichier n'a besoin d'être mis à niveau ».
/codepage: *number*	Spécifie la page de codes à utiliser pour tous les fichiers de code source inclus dans la compilation. L'outil utilise la page de codes spécifiée par *number* pour interpréter le codage des fichiers sources. L'option **/codepage** n'est pas nécessaire si les fichiers de code de source ont été enregistrés à l'aide de la page de codes ANSI actuelle, Unicode ou UTF-8 avec une signature.
/help ou **/?**	Affiche les options de l'outil.
/nobackup	Empêche l'outil de créer des fichiers de sauvegarde ; les fichiers sources existants sont toujours remplacés. Par défaut, lorsqu'un fichier est modifié, le contenu du fichier d'origine est copié dans un nouveau fichier. Le nouveau nom du fichier est dérivé en ajoutant ".old" au nom du fichier d'origine.
/nologo	Supprime l'affichage de la bannière de copyright.
/quiet	Empêche l'outil d'afficher du code pour des erreurs et des avertissements liés à la syntaxe.
/recurse: *wildcard*	Provoque le traitement par l'outil de tous les fichiers correspondant à la spécification *wildcard* dans le répertoire actif et tous les sous-répertoires. La spécification *wildcard* peut contenir un chemin d'accès pour déterminer où la récursivité commence.
/utf8output[+|-]	Affiche le résultat de l'outil dans le format de codage UTF-8. La valeur par défaut de cette option est **/utf8output-**, ce qui signifie que les résultats de la compilation n'utilisent pas le codage UTF-8. Les options **/utf8output** et **/utf8output+** sont équivalentes.
/verbose	Affiche des messages d'erreur détaillés et leur état.

Notes

Visual Basic 2005 spécifie de nouveaux mots clés réservés qui ne sont pas réservés dans Visual Basic .NET. Les fichiers sources qui utilisent les nouveaux noms de mots clés comme identificateurs ne sont pas valides dans Visual Basic 2005, bien qu'ils soient valides dans les versions antérieures du langage.

L'outil de mise à niveau de mot de clé recherche toutes les instances d'identificateurs qui représentent désormais des mots clés réservés dans la version actuelle de Visual Basic, et les place entre crochets ([ ]). Il conserve la mise en forme et n'apporte pas d'autres modifications à la source.

Attention

Utilisez cet outil uniquement sur les fichiers sources Visual Basic 7.0 ou 7.1. D'autres types de fichier peuvent être endommagés par l'addition de crochets autour de mots clés Visual Basic.

Pour mettre à niveau le fichier source et garantir sa compatibilité avec la version antérieure, placez les noms d'identificateurs entre crochets. Pour plus d'informations sur l'utilisation de mots clés comme noms d'identificateurs, consultez Utilisation des mots clés comme noms d'éléments dans le code.

Voir aussi

Référence

Syntaxe de ligne de commande de Vbupgrade

Concepts

Utilisation des mots clés comme noms d'éléments dans le code

Autres ressources

Outils de ligne de commande de Visual Basic

Mise à niveau de la référence pour Visual Basic

Cette rubrique fournit des liens vers des informations relatives à la mise à niveau de vos applications à partir de versions antérieures de Visual Basic vers la version actuelle.

Dans cette section

Référence de mise à niveau Visual Basic 6.0

Contient des informations sur les problèmes générés lors de la mise à niveau d'un projet Visual Basic 6.0 vers la nouvelle version de Visual Basic.

Rubriques connexes

Référence (Visual Basic)

Vous renvoie à des informations de référence concernant divers aspects de la programmation Visual Basic.

Aide pour les utilisateurs de Visual Basic 6.0

Vous guide tout au long de l'utilisation de vos connaissances de Visual Basic 6.0 pour être productif dans Visual Basic 2005.

Modifications du langage pour les utilisateurs de Visual Basic 6.0

Fournit des informations complètes sur les modifications apportées au langage par rapport aux versions antérieures de Visual Basic.

Informations de référence du .NET Framework

Cette rubrique vous renvoie à des informations sur l'utilisation de la bibliothèque de classes .NET Framework.

Dans cette section

Bibliothèque de classes .NET Framework dans Visual Studio

Propose des liens vers des rubriques Visual Studio qui prennent en charge le développement de fonctionnalités basées sur les classes dans le .NET Framework.

Rubriques connexes

Référence générale du .NET Framework

Contient un jeu complet de rubriques de référence pour le .NET Framework, notamment la bibliothèque de classes et la syntaxe ASP.NET.

Référence de la bibliothèque de classes .NET Framework

Contient un jeu complet de rubriques de référence pour les membres de la bibliothèque de classes du .NET Framework.

Vue d'ensemble de la bibliothèque de classes du .NET Framework

Présente les classes, les interfaces et les types valeur qui permettent d'accélérer et d'optimiser le processus de développement et de fournir l'accès aux fonctions du système.

Vue d'ensemble du .NET Framework

Contient des descriptions des principales fonctionnalités du .NET Framework, notamment le Common Language Runtime.

Exemples .NET Framework

Affiche des exemples de programmes qui montrent comment écrire des applications et des composants qui utilisent les technologies du .NET Framework.

Spécification du langage Visual Basic

Le document Spécifications du langage Visual Basic 8.0 est la source autorisée pour les réponses aux questions liées à la grammaire et à la syntaxe de Visual Basic. Il contient des informations détaillées sur tous les aspects du langage, y compris de nombreux points non couverts dans la documentation de référence Visual Basic.

La spécification Visual Basic est disponible comme fichier Microsoft Word depuis le Centre de téléchargement Microsoft. Elle l'est également au Centre pour développeurs Microsoft Visual Basic.

Voir aussi

Autres ressources

Référence Visual Basic

Exemples d'applications Visual Basic

Dans la documentation Visual Basic, un exemple est un projet Visual Basic que vous pouvez charger dans Visual Studio et exécuter. Un exemple est un petit bloc de code qui montre une tâche de programmation unique et limitée. Un extrait est un bloc de code inséré dans l'éditeur de code à l'aide de la fonctionnalité d'extrait de code IntelliSense.

En plus des exemples répertoriés ci-dessous, vous pouvez, si vous le souhaitez, en trouver d'autres sur le Centre de développement Visual Basic (http://msdn.microsoft.com/vbasic/).

Dans cette section

Exemples d'applications Visual Basic

Ces exemples illustrent les tâches du projet, de l'utilisateur et de l'assembly.

Exemples de données

Ces exemples illustrent l'accès aux données.

Exemples de langages Visual Basic

Ces exemples illustrent des concepts du langage Visual Basic.

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Exemples de composants serveur Visual Basic

Ces exemples présentent des applications qui interagissent avec les composants du système d'exploitation Windows.

Exemples de Windows Forms Visual Basic

Ces exemples illustrent des applications Windows Forms.

Comment : installer et dépanner des composants de base de données pour les exemples

Explique comment installer les exemples de base de données utilisés dans les exemples.

Conventions de codage Visual Basic

Décrit les indications de codage utilisées pour créer les exemples.

Rubriques connexes

Extraits de code IntelliSense Visual Basic

Bloc de code que vous pouvez insérer directement dans l'éditeur de code.

Exemples d'applications

Applications écrites à l'aide du .NET Framework.

Exemples de technologies

Ces exemples mettent en avant des fonctionnalités particulières du .NET Framework.

Voir aussi

Autres ressources

Guide de programmation Visual Basic

Comment : installer et dépanner des composants de base de données pour les exemples

Nombre des exemples Visual Basic utilisent la base de données Northwind qui est exécutée sur Microsoft SQL Server Express. Cette page décrit la procédure d'installation.

Pour télécharger et installer SQL Server Express

1. Dans Windows, ouvrez Internet Explorer.
2. Cliquez sur l'URL suivante : http://www.microsoft.com/downloads (Centre de téléchargement Microsoft)
3. Dans la zone **Mots clés**, tapez **SQL Server Express**.
4. Cliquez sur **Aller à**.
5. Dans la page de résultats, suivez le lien vers la page de téléchargement de SQL Server Express et conformez-vous aux instructions.

Pour installer les exemples de base de données Northwind et Pubs pour SQL Server

1. Dans Windows, ouvrez Internet Explorer.
2. Cliquez sur l'URL suivante : http://www.microsoft.com/downloads/details.aspx?FamilyID=06616212-0356-46a0-8da2-eebc53a68034&DisplayLang=en (bases de données Northwind et Pubs)
3. Cliquez sur le bouton **Télécharger**.
4. Dans la zone **Téléchargement de fichier**, choisissez **Enregistrer ce fichier maintenant** et cliquez sur **OK**.
5. Une fois le fichier téléchargé, double-cliquez sur le fichier **Nwind.exe** pour installer la base de données.

Pour installer l'exemple de base de données Northwind pour Microsoft Access

1. Dans Windows, ouvrez Internet Explorer.
2. Cliquez sur l'URL suivante : http://www.microsoft.com/downloads/details.aspx?familyid=c6661372-8dbe-422b-8676-c632d66c529c&displaylang=en (Access 2000 Tutorial: Northwind Traders Sample Database)
3. Cliquez sur le bouton **Télécharger**.
4. Dans la zone **Téléchargement de fichier**, choisissez **Enregistrer ce fichier maintenant** et cliquez sur **OK**.
5. Une fois le fichier téléchargé, double-cliquez sur le fichier **Nwind.exe** pour installer la base de données.

Pour déterminer votre chaîne de connexion

1. Ouvrez **Explorateur de serveurs**/**Explorateur de bases de données**.
2. Développez le nœud **Connexions de données** et sélectionnez la connexion voulue.
3. Connectez-vous à une base de données sur votre serveur.
4. Cliquez avec le bouton droit sur la nouvelle connexion et choisissez **Propriétés**.
5. Dans la fenêtre **Propriétés**, examinez la propriété **Chaîne de connexion**. Utilisez cette chaîne de connexion dans les exemples. La chaîne de connexion est recherchée à l'un des endroits suivants, selon l'exemple :
   - Le code dans le formulaire.
   - Le fichier app.config.
   - Les paramètres du projet. Double-cliquez sur **Mon projet** dans l'**Explorateur de solutions** et recherchez le paramètre de chaîne de connexion dans l'onglet **Paramètres**.

- Les fichiers de groupe de données.

Dépannage

- Les exemples s'attendent à ce que l'exemple de base de données soit installé sur l'ordinateur local dans l'instance VSDOTNET. Si la base de données est installée ailleurs, vous devrez mettre à jour les chaînes de connexion dans l'exemple de code.

Sécurité

Vérifiez toujours s'il existe des mises à jour du produit lors de l'installation d'un logiciel supplémentaire.

Voir aussi

Tâches

Comment : installer des exemples de bases de données

Autres ressources

Connexion aux données avec l'Explorateur de serveurs/Explorateur de bases de données

Exemples d'applications Visual Basic

Ces exemples illustrent les tâches de l'architecture, du projet, de l'utilisateur et de l'assembly.

Dans cette section

Événements d'application, exemple

Montre les événements d'application accessibles par l'intermédiaire du **Concepteur de projets**.

ClickOnce, exemple

Déploie et met à jour une application à l'aide de ClickOnce.

Application console, exemple

Illustre une entrée et une sortie simples à l'aide de la fenêtre de console.

Consommation d'un service Web, exemple

Montre comment utiliser différents services Web .NET Framework existants, notamment les services qui fournissent des informations météorologiques, un dessin animé quotidien et un outil de conversion des devises.

Jeu, exemple

Illustre la fonctionnalité **My**, la programmation orientée objet et le dessin.

Garbage Collection, exemple

Montre comment contrôler le garbage collector du CLR (Common Language Runtime).

Enregistrement dans un journal, exemple

Utilise l'objet **My.Application.Log** pour écrire des informations dans les fichiers et les journaux des événements.

Multithreading, exemple

Montre deux façons différentes d'implémenter des tâches multithread.

Office Automation, exemple

Présente l'automatisation de Microsoft Agent, de Microsoft Word et de Microsoft Excel.

Réflexion, exemple

Utilise la réflexion pour découvrir des informations de type dans un assembly.

Ressources et paramètres, exemple

Illustre les objets **My.Settings** et **My.Resources**.

Envoi de courrier, exemple

Montre comment envoyer du courrier électronique via SMTP à l'aide de classes dans l'espace de noms System.Web.Mail.

Inspection du frame de pile, exemple

Montre comment récupérer par programme des informations sur la pile des appels à tout moment depuis le code en cours d'exécution.

Informations système et d'environnement, exemple

Présente l'utilisation de plusieurs classes qui fournissent des informations d'environnement.

Accès distant TCP, exemple

Montre comment utiliser l'architecture d'accès distant .NET Framework.

Informations utilisateur, exemple

Fournit une vue d'ensemble de l'objet **My.User** et du modèle **Formulaire de connexion**. Il implémente une authentification personnalisée simple.

Mise à niveau d'une application Visual Basic 6.0, exemple

Présente l'exemple de magnétoscope Visual Basic 6.0 dans le code Visual Basic 6.0 et Visual Basic 2005.

Référence

Référence My

La fonctionnalité **My** fournit l'accès aux propriétés d'application et d'utilisateur.

My.Application, objet

L'objet **My.Application** fournit l'accès aux propriétés d'application.

Rubriques connexes

Exemples de données

Ces exemples illustrent des tâches de manipulation de données.

Exemples de Windows Forms Visual Basic

Ces exemples illustrent des applications Windows Forms.

Exemples de langages Visual Basic

Ces exemples illustrent des concepts du langage Visual Basic.

Exemples de composants serveur Visual Basic

Ces exemples illustrent des applications qui interagissent avec les composants du système d'exploitation Windows.

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Exemples d'applications Visual Basic

Cette page est le point de début pour tous les exemples Visual Basic inclus avec la documentation.

Événements d'application, exemple

Download sample

Cet exemple montre comment utiliser des événements **My.Computer.Network** pour écrire des applications orientées réseau.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

`MainForm` contient un contrôle StatusStrip qui indique si la connexion réseau est actuellement connectée ou déconnectée. Chaque fois que l'état de connectivité réseau change, l'événement My.Computer.Network.NetworkAvailabilityChanged est déclenché. Le gestionnaire de cet événement réside dans le fichier Mon projet\MyEvents.vb. Pour accéder à ce fichier, double-cliquez sur **Mon projet** dans l'**Explorateur de solutions**. Sur l'onglet **Application** du **Concepteur de projets**, cliquez sur le bouton **Afficher le code**. La logique qui met à jour le volet **État** réside dans la méthode `SetConnectionStatus` de `MainForm`. Les événements réseau en Visual Basic sont marshalés jusqu'au thread d'application principal ; les gestionnaires de ces événements peuvent, par conséquent, manipuler directement l'interface utilisateur.

Voir aussi

Référence

My.Computer.Network, objet

ClickOnce, exemple

Download sample

Cet exemple illustre plusieurs tâches qui utilisent la fonctionnalité ClickOnce.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour générer et publier cette solution

1. Générez la solution : cliquez avec le bouton droit sur le nom du projet dans l'**Explorateur de solutions** et choisissez **Générer**.

2. Publiez la solution : cliquez avec le bouton droit sur le nom du projet dans l'**Explorateur de solutions** et choisissez **Publier**. Pour plus d'informations sur l'Assistant Publication, consultez Comment : publier une application ClickOnce.

Démonstrations

La fonctionnalité ClickOnce peut être utilisée pour gérer par programme plusieurs installations et tâches de maintenance, y compris :

- **La vérification des mises à jour** Utilise la méthode CheckForUpdate pour vérifier les nouvelles mises à jour.
- **La mise à jour d'une application** Utilise la méthode UpdateAsync pour télécharger l'application mise à jour.
- **Le téléchargement de fichiers facultatifs** Utilise la méthode DownloadFileGroupAsync pour télécharger les fichiers.
- **Les informations de déploiement** La propriété **My.Application.Deployment** est utilisée pour récupérer et rapporter le nom, la version, l'emplacement et la dernière vérification de la mise à jour.

Voir aussi

Référence

My.Application.Deployment, propriété
My.Application.IsNetworkDeployed, propriété
CheckForUpdateCompleted
DownloadFileGroupCompleted
UpdateProgressChanged
UpdateCompleted

Autres ressources

Déploiement ClickOnce

Application console, exemple

Download sample

Cet exemple montre comment créer des applications console.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le projet inclut un fichier XML comme une ressource imbriquée. Le fichier XML contient la table `Products` de la base de données Northwind. L'application charge un groupe de données non typé à partir du fichier XML. Lorsque l'application console est exécutée, elle accepte un ID de produit ou "quit" comme entrée de l'utilisateur. Si "quit" est entré, l'application console se termine. Si un ID de produit valide est entré, les informations associées au produit sont affichées. Si un ID de produit non valide est entré, un message d'exception est affiché.

Voir aussi

Référence

Console Class
Structure d'un programme Visual Basic
DataSet.ReadXml Method

Concepts

Génération d'applications console

Consommation d'un service Web, exemple

Download sample

Cette application montre comment utiliser différents services Web .NET Framework existants, notamment les services qui fournissent des informations météorologiques, un dessin animé du jour et un outil de conversion des devises.

Un TabControl fournit cinq exemples différents d'utilisation de services Web.

- Obtenez la date et l'heure locale en fonction d'un code postal américain (http://www.ripedev.com/webservices/LocalTime.asmx).
- Convertissez l'euro/dollar dans l'une des monnaies des membres de l'ECC (http://www.xml-webservices.net/services/conversions/euro_convert/euro_conver.asmx).
- Recherchez des classements de vente et des informations sur le prix d'un livre disponible sur Amazone et Barnes & Noble en entrant un numéro ISBN (http://www.perfectxml.net/WebServices/SalesRankNPrice/BookService.asmx).
- Obtenez les conditions météorologiques actuelles en fonction d'un code postal américain (http://www.learnxmlws.com/services/weatherRetriever.asmx).
- Visionnez le dessin animé Dilbert du jour à partir du service Web quotidien Dilbert. Cet exemple appelle le service Web de façon asynchrone (http://www.esynaps.com/WebServices/DailyDilbert.asmx).
- Consultez comment l'UDDI peut être utilisé pour gérer des erreurs d'URL de service Web, en vous permettant d'implémenter un "plan de secours" (http://www.vbws.com/services/ServerTime.asmx). les messages d'état et les messages d'erreur dans la gestion structurée des erreurs sont utilisés pour informer l'utilisateur pendant le processus de récupération des données.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cette application requiert le UDDI (Microsoft Universal Description Discovery and Integration) SDK 1.76, qui peut être téléchargé à partir du site suivant : http://msdn.microsoft.com/library/en-us/dnanchor/html/anch_uddi.asp?frame=true

Démonstrations

Ces services Web réels ont été trouvés sur le site http://www.xmethods.com/.

Voir aussi

Tâches

Comment : accéder à un service Web XML en code managé

Jeu, exemple

Download sample

Cet exemple d'application implémente un jeu Windows Forms, complet avec des graphiques GDI+, des fonctionnalités de minuterie, une configuration utilisateur et un stockage des meilleurs scores.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Sélectionnez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

Cet exemple illustre les fonctionnalités qui sont implémentées dans la plupart des applications :

- **Options utilisateur** Le formulaire `Options` permet aux utilisateurs de réinitialiser les meilleurs scores et d'activer et de désactiver le son.

- **Aide** L'application inclut un projet d'aide compilé. L'aide compilée est accessible via l'élément de menu **?** (Aide).

- **Souris** La souris est utilisée non seulement pour jouer au jeu, mais aussi pour déplacer le formulaire lorsque la barre de titre est masquée.

- **Clavier** La touche "M" peut activer et désactiver le menu et la barre de titre. La touche "P" permet de suspendre et de redémarrer l'action de jeu.

Plusieurs fonctionnalités Visual Basic et .NET Framework sont mises en surbrillance :

- **Graphiques GDI+** L'action de jeu est complètement implémentée à l'aide de fonctions de dessin de l'espace de noms System.Drawing.

- **Registre** Les meilleurs scores sont stockés dans le Registre.
- **Programmation orientée objet** L'action de jeu et le dessin sont contrôlés par plusieurs classes agissant ensemble. Les classes de jeu principales sont les classes `Grid` et `Block`. La classe `PointTranslator` ajoute des fonctions utilitaires pour dessiner les blocs. La classe `HighScores` gère la récupération et la définition des scores.

Voir aussi

Référence

My.Computer.Audio, objet
My.Computer.Registry, objet
System.Drawing
HelpProvider
MouseDown
KeyDown

Garbage Collection, exemple

Download sample

Cet exemple montre comment contrôler le garbage collector du Common Language Runtime (CLR). Il illustre le moment où le garbage collector appelle une méthode `Finalize` sur des objets créés ainsi que comment supprimer la finalisation.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5.

2. Fermez l'application et remarquez comment tout objet non finalisé est finalisé à la fermeture du formulaire.

Démonstrations

Les boutons sur le formulaire exécutent le code qui vous permet de manipuler le nombre d'objets alloués, d'objets libres et d'objets récupérés par le garbage collector.

- **Create Objects** Ce bouton crée une hiérarchie d'objets du type `ListNode` qui est définie dans le projet.

- **Set Object to Nothing** Ce bouton libère l'objet sélectionné dans la zone de liste `lstCreatedObjects` pour une opération garbage collection en lui affectant la valeur **Nothing**.

- **Call Dispose** Ce bouton se débarrasse de l'objet sélectionné dans la zone de liste `lstCreatedObjects`. L'interface IDisposable est implémentée sur la classe `ListNode`.

- **Run Garbage Collector** Cette sous-routine force le garbage collector à s'exécuter. La méthode Collect n'est pas déterministe. Cela signifie que le système d'exploitation peut différer l'exécution du garbage collector pour terminer des tâches en cours d'exécution. En pratique, vous pouvez vous attendre à ce que le garbage collector exécute en une seconde ou presque la méthode `Collect` qui est appelée.

L'exemple montre également comment la performance du garbage collector est améliorée sur certains objets à l'aide de la méthode SuppressFinalize.

Voir aussi

Référence

GC
Collect

SuppressFinalize

Autres ressources

Garbage collection

Enregistrement dans un journal, exemple

Download sample

Cet exemple montre comment utiliser et configurer l'objet **My.Application.Log**.

Visual Basic offre deux mécanismes d'enregistrement différents. **My.Application.Log** fournit l'accès aux journaux des événements du système d'exploitation et vous permet également d'écrire des messages dans des fichiers texte. Le composant EventLog fournit l'accès aux journaux des événements du système d'exploitation. Le composant **EventLog** vous permet d'ajouter et de supprimer des journaux, d'ajouter et de supprimer des sources de journal des événements, d'écrire et de supprimer des messages. Pour obtenir un exemple de code, consultez Journal des événements, exemple.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5. Le formulaire principal répertorie les écouteurs qui sont configurés pour l'exemple.
2. Écrivez un message dans le contrôle TextBox sur le formulaire.
3. Cliquez sur le bouton **Write Log Message**.

   Le message est écrit dans chacun des écouteurs.

   a. Le chemin d'accès au fichier XML écrit par XmlWriterTraceListener est c:\logsamples\SampleLog.xml.

   b. Le fichier texte simple écrit par FileLogTraceListener se trouve dans votre répertoire "Application Data". Si vous exécutez le programme dans Visual Studio, le chemin d'accès est du type "c:\Documents And Settings\[user]\Application Data\Microsoft Corporation\Microsoft Visual Studio \[8.0*] \LogSample.log".

Démonstrations

Cet exemple contient :

- `Form1` Fournit un formulaire simple qui affiche tous les écouteurs actuellement activés pour **My.Application.Log** et dispose de contrôles qui permettent que du texte soit écrit dans **My.Application.Log**.
- `app.config` Définit la configuration pour **My.Application.Log** (où les messages doivent être écrits, quels sont les messages qui doivent être filtrés, etc.).

Cet exemple montre comment **My.Application.Log** peut être utilisé pour écrire un fichier texte simple, le journal des événements et un fichier XML. Vous pouvez implémenter facilement un TraceListener personnalisé qui écrit également dans

d'autres emplacements, tels qu'une base de données ou un message électronique.

La "Gravité" sélectionnée sur le formulaire est assignée à chaque message. Lorsque l'écouteur reçoit le message, il peut ignorer le message selon son niveau de filtre. Par exemple, le composant EventLogTraceListener est configuré uniquement pour écrire des messages d'erreur ; par conséquent, tous les messages que vous écrivez avec un niveau de gravité "Information" ou "Avertissement" sont écrits dans le fichier XML et dans le fichier texte brut, et pas dans **EventLog**.

Pour modifier le comportement de l'exemple, modifiez le fichier app.config du projet. Les commentaires dans le fichier servent de guide pour les modifications.

Vous pouvez utiliser **My.Application.Log** et le modèle d'application pour enregistrer des exceptions de démarrage, des exceptions d'arrêt et des exceptions non gérées. Pour consulter le code pour cet exemple, double-cliquez sur **Mon projet** dans l'**Explorateur de solutions**. Dans le volet **Application**, cliquez sur le bouton **Afficher le code**.

Voir aussi

Concepts

Utilisation des journaux d'applications dans Visual Basic

Autres ressources

Journal des événements, exemple

Multithreading, exemple

Download sample

Cette solution montre comment utiliser plusieurs threads pour obtenir l'accès concurrentiel dans les tâches.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple exécute une tâche nécessitant du temps sur un thread séparé, lancée en cliquant sur un bouton :

- **Exécution sur le même thread** La tâche s'exécute sur le même thread que le formulaire principal. Cela empêche l'utilisateur d'interagir avec le formulaire principal avant que la tâche ne soit terminée. Aucun code multithread n'est nécessaire pour cette tâche.

- **Exécution sur un pool de threads de travail** La tâche utilise la méthode `BeginInvoke` créée par le compilateur pour que tous les délégués exécutent la tâche.

- **Travail d'arrière-plan** Cette tâche utilise le composant BackgroundWorker pour créer un nouveau thread.

Cet exemple n'inclut pas de synchronisation, parce que les threads multiples n'accèdent à aucune donnée.

Voir aussi

Référence

Delegate, instruction
DebuggerStepThroughAttribute Class
AppDomain.GetCurrentThreadId Method
Thread Class

Concepts

Vue d'ensemble de la programmation asynchrone

Autres ressources

Programmation asynchrone à l'aide de délégués

Office Automation, exemple

Download sample

Cet exemple illustre l'automation de Microsoft Agent, Microsoft Word et Microsoft Excel. Lorsque vous référencez les différents assemblys COM Office, Visual Studio crée automatiquement des wrappers pouvant être appelés par le runtime et par COM (RCW, Runtime Callable Wrappers) afin que vous puissiez programmer en vous servant d'eux à partir du .NET Framework.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

6. En fonction de la configuration de votre installation Office, les références Office peuvent ne pas être valides dans le projet. Pour mettre à jour les références :

   a. Double-cliquez sur **Mon projet** dans l'**Explorateur de solutions**.

   b. Sélectionnez l'onglet **Références** dans le **Concepteur de projets**.

   c. Cliquez sur **Ajouter**.

   d. Dans la boîte de dialogue **Ajouter une référence**, cliquez sur l'onglet **COM**.

   e. Recherchez **Microsoft Agent Control**, puis double-cliquez dessus pour l'ajouter à votre projet.

   f. Répétez la procédure pour ajouter les références pour Microsoft Word et Microsoft Excel.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Si vous souhaitez entendre Merlin parler, vous devrez peut-être installer le pack de reconnaissance vocale qui est inclus avec Microsoft Office XP, mais qui n'est pas installé par défaut. Pour obtenir des instructions à ce sujet, consultez : Install speech recognition http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/input_speech_install.mspx).

Démonstrations

Trois contrôles TabPage sont ajoutés à un TabControl pour illustrer trois tâches de programmation :

- **Microsoft Agent** La bibliothèque Office Agent fait apparaître le personnage de Merlin et l'exécute. Avant que le personnage de Merlin ne puisse apparaître, une instance de l'objet `AgentObjects.Controller` doit être créée. Un objet `IAgentCtlCharacter` qui représente Merlin est accessible à partir de cet objet. `IAgentCtlCharacter` implémente une méthode `Play` qui est utilisée pour contrôler les différentes animations. Pour plus d'informations sur Microsoft Agent,

consultez les articles et la documentation du Kit de développement à l'adresse http://www.microsoft.com/france/msdn.

- **Microsoft Word** Un fichier texte simple est affiché dans un contrôle RichTextBox. Une instance de Word est démarrée et l'application Word est utilisée pour vérifier l'orthographe du texte dans le contrôle **RichTextBox**.

- **Microsoft Excel** Créez et remplissez un DataSet à partir d'un document XML, puis utilisez la méthode DataBind pour le lier à un DataGrid. Exportez ensuite le contenu de **DataSet** vers une feuille de calcul Excel, puis exécutez une fonction Excel qui calcule la moyenne des valeurs dans une colonne.

Voir aussi

Tâches

Comment : installer les assemblys PIA (Primary Interop Assembly) d'Office

Comment : cibler les applications Office via les assemblys PIA

Concepts

Vue d'ensemble des assemblys dans les projets Office

Réflexion, exemple

Download sample

Cet exemple utilise la réflexion pour découvrir des informations de type dans un assembly.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le formulaire vous permet d'examiner les assemblys à trois niveaux :

- **Assembly** La méthode GetAssemblies est utilisée pour récupérer une liste des assemblys chargés. La liste est affichée dans un contrôle ListBox.
- **Type** La sélection d'un nom d'assembly affiche tous les types dans cet assembly dans un autre contrôle **ListBox**, à l'aide de la méthode GetTypes.
- **Membres** Lorsqu'un type est sélectionné dans la zone de liste **Types**, la méthode GetMembers est utilisée pour répertorier tous les membres du type sélectionné.

Voir aussi

Référence

Assembly
AppDomain
GetAssemblies
GetTypes
GetMembers

Concepts

Vue d'ensemble de la réflexion

Ressources et paramètres, exemple

Download sample

Cet exemple montre comment gérer des ressources et des paramètres pour une application à l'aide de l'objet **My.Settings**, de l'objet **My.Resources**, du **Concepteur de projets** et de l'**Éditeur de ressources**.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

  Chaque fois que vous lancez l'application, un arrière-plan différent est affiché.

Démonstrations

Un paramètre est utilisé pour effectuer le suivi du nombre de fois où l'application a été lancée. Chaque fois que l'application s'exécute, le paramètre est examiné et l'une des nombreuses images de bureau disponibles est sélectionnée selon ce nombre. Par conséquent, chaque fois que vous lancez l'application, vous voyez un nouvel arrière-plan.

Le code réside dans la méthode `SetBackground` de `Form1` et dans les événements de démarrage et d'arrêt de l'application qui sont gérés dans Mon projet\MyEvents.vb. Pour afficher ce fichier, double-cliquez sur **Mon projet** dans l'**Explorateur de solutions**. Dans le volet **Application** du **Concepteur de projets**, cliquez sur le bouton **Afficher les événements de l'application**.

Des paramètres sont créés et modifiés sous l'onglet **Paramètres** du **Concepteur de projets**. Il existe deux types de paramètres : les paramètres d'application et les paramètres utilisateur. Les paramètres d'application sont ceux qui spécifient des valeurs communes à toutes les instances de l'application. Les paramètres utilisateur sont créés et modifiés pour chaque utilisateur. Dans cet exemple, le paramètre qui rend persistant le nombre de fois où une installation de l'application a été exécutée est un paramètre utilisateur.

Voir aussi

Concepts

Accès aux ressources de l'application
Accès aux paramètres d'application

Envoi de courrier, exemple

Download sample

Cet exemple montre comment envoyer le courrier électronique sur SMTP à l'aide de classes dans l'espace de noms System.Web.Mail.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Vérifiez le paramètre de propriété SmtpServer. Pour plus d'informations, consultez « Configuration requise » ci-dessous.

2. Appuyez sur F5.

Configuration requise

1. Les services IIS (Internet Information Services) sont requis.

2. Le Service SMTP doit être installé et en cours d'exécution. Le service SMTP est une option installée avec les services IIS.

3. La propriété **SmtpServer** doit avoir pour valeur l'adresse IP ou le nom de serveur où le Serveur SMTP s'exécute. Dans la plupart des cas c'est l'ordinateur "localhost". En tant que tel, la valeur de propriété **SmtpServer** peut être le nom convivial, "localhost" ou "127.0.0.1". Ce dernier est la valeur par défaut pour cette application. Pour rechercher le nom de Serveur SMTP :

   a. Ouvrez le composant logiciel enfichable IIS (Dans **Panneau de configuration**, ouvrez **Outils d'administration**, puis ouvrez **Services IIS**).

   b. Développez le nœud **Default SMTP Virtual Server**.

   c. Cliquez sur **Domaines**. Dans le volet droit, une liste de **Noms de domaines** s'affiche. Vous pouvez affecter un de ces noms ou l'adresse IP correspondante à la propriété **SmtpServer**.

4. Le serveur virtuel SMTP par défaut doit également être en cours d'exécution. Sachez que le Service SMTP peut être en cours d'exécution, mais le serveur virtuel SMTP par défaut dans les services IIS ne doit pas être en cours d'exécution. Il y a une différence entre le service lui-même et le serveur réel exécuté par le service. Si votre serveur virtuel SMTP par défaut ne s'exécute pas, une icône X rouge sera visible sur son nœud dans le composant logiciel enfichable IIS. Pour démarrer le serveur, cliquez avec le bouton droit sur son nœud et cliquez sur **Démarrer**.

5. Définissez les restrictions de relais :

    a. Dans les services IIS, cliquez avec le bouton droit sur **Serveur virtuel SMTP par défaut** et sélectionnez **Propriétés**.

    b. Dans la boîte de dialogue **Propriétés**, cliquez sur l'onglet **Accès**.

    c. Cliquez sur **Relais**.

    d. Dans la boîte de dialogue **Restrictions de relais**, sélectionnez la zone intitulée **Tous sauf la liste ci-dessous**. La case à cocher située en bas peut être activée ou désactivée.

Démonstrations

Cette application possède un formulaire seul avec des contrôles semblable au client de messagerie électronique Microsoft Outlook. Lorsque le formulaire se charge, il vérifie que l'utilisateur possède le Service SMTP installé et que celui-ci est en cours d'exécution.

Voir aussi

Tâches

Mailer, exemple : illustre des classes de courrier SMTP

Comment : installer et configurer des serveurs virtuels SMTP dans IIS

Inspection du frame de pile, exemple

Download sample

Cet exemple montre comment récupérer par programme des informations sur la pile des appels à tout point du code exécutant. La pile vous permet de déterminer la séquence des procédures que l'application a exécutée avant d'arriver à la procédure actuelle.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le .NET Framework fournit la fonctionnalité en tant qu'élément de la classe StackFrame. L'exemple illustre deux autres classes : StackTrace et MethodInfo. Le constructeur sans paramètre de **StackTrace** retourne un jeu complet d'objets **StackFrame**, fournissant des informations provenant du démarrage de votre application. Vous pouvez également utiliser le constructeur **StackTrace** qui prend une instance de Exception. Ce constructeur renvoie des informations de traçage de la pile au code qui a déclenché l'exception.

Le gestionnaire d'événements pour le bouton **Test Procedure Stack** contient le code pour appeler plusieurs méthodes qui résultent en une pile de plusieurs niveaux. Cliquer sur le bouton **Test Procedure Stack** appelle **ProcA**, qui appelle **ProcB**. **ProcB** appelle la procédure **GetFullStackFrameInfo**, en passant dans un nouvel objet `StackTrace` :

```
GetFullStackFrameInfo(New StackTrace())
```

Dans la procédure `GetFullStackFrameInfo`, le code utilise la propriété FrameCount de l'objet **StackTrace** pour récupérer le nombre total de frames de pile, en travaillant antérieurement à la procédure actuelle. La procédure ajoute ensuite des informations concernant chaque frame de pile à la zone de liste de l'exemple, en appelant la méthode GetFrame pour récupérer un objet de frame de pile indexé.

La procédure **GetStackFrameInfo** appelle **GetFrame** sur l'objet **StackFrame** que le code lui a envoyé, puis récupère les propriétés de l'objet **MethodInfo** pour afficher les informations relatives à la procédure dans la zone de liste.

Le gestionnaire d'événements pour le bouton **Test Exception Handling** génère une erreur en bas de la pile de plusieurs couches. Le code déclenche une exception et passe cette exception au constructeur pour l'objet **StackFrame**.

Voir aussi

Référence
System.Diagnostics
StackFrame
System.Reflection
MethodInfo
FrameCount

Informations système et d'environnement, exemple

Download sample

Cet exemple illustre l'utilisation de plusieurs classes qui fournissent des informations d'environnement.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le formulaire principal contient un TabControl avec plusieurs pages d'onglets. Chaque page d'onglets appelle un groupe différent de membres Environment ou SystemInformation :

- **Propriétés** Cet onglet présente les propriétés **Environment**.
- **Dossiers spéciaux** Le code appelle la méthode GetNames sur SpecialFolder pour récupérer une liste de tous les noms de dossiers spéciaux. Cette liste est utilisée comme source de données pour le contrôle ListBox.
- **Méthodes** Cet onglet affiche les résultats de l'appel aux méthodes **Environment**, y compris Exit et ExpandEnvironmentVariables. Les tableaux de chaînes retournés par les méthodes GetLogicalDrives et GetCommandLineArgs sont utilisés comme sources de données pour les contrôles **ListBox**.
- **Variables d'environnement** Le code appelle la méthode GetEnvironmentVariables. La propriété `Keys` de la collection retournée est utilisée comme source de données pour un contrôle **ListBox**. Cette page d'onglets présente également la récupération d'une variable d'environnement en appelant la méthode GetEnvironmentVariable.
- **Informations système** Cet onglet récupère toutes les informations fournies par **SystemInformation**.

Voir aussi

Référence

Environment
SystemInformation

Accès distant TCP, exemple

Download sample

Cet exemple montre comment utiliser l'architecture .NET Framework Remoting. De plus, il utilise des fichiers de configuration XML pour spécifier des objets disponibles et des informations sur l'emplacement serveur au lieu de les coder en dur dans les binaires ou de mettre les données dans la base de registres.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Générez le projet `RemoteCustomer`.

2. Ouvrez le projet `RemoteHost`, puis appuyez sur F5 pour démarrer l'application `RemoteHost`.

3. Lancez une autre instance de Visual Studio et chargez le projet `Client`, puis ajoutez une référence à l'application `RemoteCustomer` compilée (`RemoteCustomer.dll`).

4. Appuyez sur F5 pour démarrer l'application `Client`.

5. Pour tester sur deux autres ordinateurs, vous devrez apporter des modifications au fichier .config du client, en modifiant `localhost` en adresse IP ou en nom DNS pour l'ordinateur serveur. De plus, vous devrez peut-être modifier le fichier du serveur si vous modifiez le numéro de port. Consultez les commentaires dans le fichier .config du client et le fichier .config de l'hôte pour les exemples.

Démonstrations

L'exemple contient trois solutions, chacune avec un projet:

- `RemoteCustomer` : le serveur
- `RemoteHost` : l'hôte
- `Client` : le client

RemoteCustomer

`RemoteCustomer` est une bibliothèque de classes qui expose trois types pouvant être créés :

- `Customer` (activé par le client)

- `SingleCallCustomer` (singlecall WellKnown)
- `SingletonCustomer` (singleton WellKnown)

Chaque classe vit dans le même espace de noms, `RemotingSample`, qui est défini dans chaque fichier source de classe.

RemoteHost

Le projet `RemoteHost` contient une application Windows Forms qui lira un fichier de configuration XML nommé Host.exe.config afin de rendre disponible les classes définies dans le projet `RemoteCustomer`. Pour une application de production, vous pouvez souhaiter qu'un service Windows héberge la bibliothèque de classes (RemoteCustomer.dll).

Client

Le projet `Client` est une application Windows Forms. Il s'appuie sur un fichier de configuration nommé Client.exe.config pour savoir où rechercher les objets distants.

Voir aussi

Référence

System.Runtime.Remoting

Autres ressources

Vue d'ensemble de .NET Framework Remoting

Informations utilisateur, exemple

Download sample

Cet exemple fournit une vue d'ensemble de l'objet **My.User** et du modèle **Formulaire de connexion**. Il implémente une authentification personnalisée simple.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple illustre les tâches suivantes :

- **Nom d'utilisateur** Le nom d'utilisateur Windows actuel est affiché sur le formulaire principal. Cette valeur est récupérée de l'objet My.User.

- **Accès aux bases de données** **My.User** autorise des développeurs à brancher leurs propres moyens d'authentification et d'autorisation personnalisées par l'intermédiaire des interfaces IPrincipal et IIdentity. Ces interfaces décrivent des aspects de l'utilisateur actuel qui sont indépendants d'une architecture d'authentification et d'autorisation spécifique. Les implémenteurs des interfaces définissent la façon dont l'utilisateur doit être authentifié et les rôles dans lesquels l'utilisateur est actuellement inscrit. Cet exemple fournit une implémentation très basique des interfaces **IPrincipal** et **IIdentity** pour montrer comment **My.User** peut être configuré pour utiliser respectivement des mécanismes d'autorisation et d'authentification définis par le développeur.

Remarque de sécurité
Dans cet exemple, l'implémentation ne présente pas de pratiques de codage sécurisé. Si l'utilisateur tape "mot de passe" pour le mot de passe, l'utilisateur est authentifié.

- **Mode d'authentification** Le comportement du programme est contrôlé par le mode d'authentification. Ouvrez le **Concepteur de projets** en double-cliquant sur le nœud **Mon projet** dans l'**Explorateur de solutions**. Sous l'onglet **Application**, **Défini au niveau de l'application** est sélectionné pour le **Mode d'authentification**. Lorsque vous lancez l'application, le formulaire **Connexion** est affiché.

Voir aussi

Référence

My.User, objet
IPrincipal
IIdentity

Mise à niveau d'une application Visual Basic 6.0, exemple

Download sample

Cet exemple illustre une application dans Visual Basic 6.0 et montre comment cette application peut être mise à niveau vers Visual Basic 2005.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

6. Ouvrez le projet Visual Basic 6.0 dans Visual Basic 6.0.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Exemples de modifications :

- **Approche orientée objet** Les classes `Tape`, `Recorder`, `Channel` et `TapeFrame` sont implémentées. Les modules sont remplacés par les classes.
- **ByRef et ByVal** **ByRef** est remplacé par **ByVal** à l'endroit approprié.
- **Commentaires XML** Ces commentaires remplacent les blocs de commentaires mis en forme.
- **Conventions d'affectation de noms** Les noms hongrois sont supprimés.
- **Types Boolean** Les variables **Boolean** remplacent les valeurs **Integer**.

Voir aussi

Autres ressources

Exemples d'applications Visual Basic

Exemples de données

Ces exemples illustrent plusieurs scénarios de données.

Dans cette section

Contrôles de données personnalisés, exemple

Montre comment utiliser vos propres contrôles avec la fenêtre **Source de données**.

Liaison de données à un contrôle ComboBox, exemple

Montre comment lier des données à un contrôle ComboBox.

Création de base de données, exemple

Les instructions SQL sont exécutées pour créer une base de données, une table, une procédure stockée et un affichage dans le code.

Recherche de données, exemple

Montre comment afficher un sous-ensemble de données à partir de votre base de données.

Validation des données, exemple

Illustre le **Concepteur de sources de données** et utilise les événements `PropertyChanged` pour valider l'entrée d'utilisateur.

Données locales, exemple

Illustre les fonctionnalités de données client de Visual Studio 2005, y compris la fenêtre **Sources de données** et les objets TableAdapter et BindingSource typés.

Liaison d'objets, exemple

Montre comment vous pouvez lier des contrôles à des objets autres que des composants de base de données.

Sérialisation, exemple

Montre comment sérialiser et désérialiser un objet.

Procédure stockée, exemple

Montre comment créer et utiliser par programme des procédures stockées avec ADO.NET et Microsoft SQL Server.

Classe XMLDocument, exemple

Montre comment utiliser la classe XmlDocument pour manipuler des données XML.

Référence

Création d'applications de données clientes

Décrit les outils disponibles dans Visual Studio pour créer des applications de base de données, ainsi que pour concevoir et maintenir des bases de données.

System.Data

Décrit les objets .NET Framework couramment utilisés dans les applications de données.

XmlDocument

Décrit la classe utilisée pour manipuler des données XML.

Rubriques connexes

Composant BindingSource, exemple

Illustre la liaison vers les contrôles Windows Forms.

Création d'applications de données clientes

Présente la programmation de l'accès aux données dans Visual Studio.

Utilisation de données XML

Présente les outils et fonctionnalités Visual Studio permettant de travailler avec les données XML, les données XSLT et les schémas.

Exemples d'applications Visual Basic

Illustre les tâches du projet, de l'utilisateur et de l'assembly.

Exemples de Windows Forms Visual Basic

Illustre les applications Windows Forms.

Exemples de langages Visual Basic

Illustre les concepts du langage Visual Basic.

Exemples de composants serveur Visual Basic

Illustre des applications qui interagissent avec les composants du système d'exploitation Windows.

Exemples de sécurité Visual Basic

Illustre les tâches de sécurité.

Exemples d'applications Visual Basic

Donne une liste complète d'exemples Visual Basic.

Contrôles de données personnalisés, exemple

Download sample

Cet exemple montre comment utiliser vos propres contrôles avec la fenêtre **Sources de données**.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cet exemple requiert l'exemple de base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

La solution de l'exemple inclut deux projets. L'un est un projet **Bibliothèque de contrôles Windows** nommé `MyCompanyControls`. L'autre est un projet **Application Windows** nommé `CustomDataControls`.

La bibliothèque de contrôles définit le contrôle `PhoneBox`. Le contrôle hérite simplement du contrôle MaskedTextBox et affecte un numéro de téléphone avec indicatif régional à la propriété Mask. Le projet de bibliothèque contient également une ressource incorporée, PhoneBox.bmp, pour représenter l'icône des fenêtres **Boîte à outils** et **Sources de données**.

Le contrôle `PhoneBox` est disponible dans le projet `CustomDataControls`. En sélectionnant **Régénérer** dans le menu contextuel du projet, le contrôle `PhoneBox` apparaîtra dans la **boîte à outils** dans une nouvelle section nommée **MyCompanyControls**. `Form1` doit être ouvert dans le concepteur pour pouvoir consulter les éléments de la **boîte à outils**.

La fenêtre **Sources de données** est utilisée pour ajouter `PhoneBox` à la liste de contrôles pour **String**. Dans la fenêtre **Sources de données**, sélectionnez la colonne `HomePhone` sous la table `Employee`. Sélectionnez la liste déroulante et choisissez **Personnaliser**. La boîte de dialogue destinée à personnaliser la liste de contrôles de types donnés s'ouvre. Cette liste est remplie avec la liste de contrôles actuellement dans la **boîte à outils** Visual Studio qui implémente DefaultBindingPropertyAttribute. Le contrôle `PhoneBox` hérite de **MaskedTextBox** qui prend en charge **DefaultBindingPropertyAttribute**. Sélectionnez **Chaîne** sous **Type de données** et recherchez le contrôle `PhoneBox` dans la liste. La vérification par le contrôle `PhoneBox` indique à Visual Studio que le contrôle `PhoneBox` est l'un des contrôles disponibles pour tout objet qui possède une propriété de type chaîne.

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples

Comment : ajouter des contrôles personnalisés à la fenêtre Sources de données

Autres ressources

Exemples d'applications Visual Basic Express

Liaison de données à un contrôle ComboBox, exemple

Download sample

Cet exemple montre comment lier des données aux contrôles ComboBox et DataGridView.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cet exemple requiert l'exemple de base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

Le code montre comment lier six types différents de sources de données à un contrôle **ComboBox**. Les données sont liées à partir de :

- un tableau
- ArrayList de chaînes
- **ArrayList** d'objets de classe
- DataTable
- DataView
- Objet BindingSource.

Lorsque le formulaire principal se charge, la table `Products` de la base de données Northwind est récupérée dans un DataSet à l'aide d'une simple instruction SQL Select. **DataView** qui fournit une vue triée de la colonne `ProductName` est également créé à ce moment. La table `Products` est chargée dans un autre **DataSet** à l'aide d'un TableAdapter et d'un **BindingSource**.

L'utilisateur peut alors remplir le contrôle zone de liste déroulante en liant un autre tableau de couleurs, une liste de tableaux de formes, une liste avancée de tables contenant les départements des ventes définis avec une structure, la table de produits qui réside dans l'un des groupes de données ou la vue triée de données. Si l'utilisateur lie le groupe de données, la vue de données ou la liste avancée de tables de départements des ventes, lorsqu'une entrée est sélectionnée à partir de la zone de

liste déroulante, une valeur associée pour cette entrée est également affichée. Si l'utilisateur établit une liaison au connecteur de données, la zone de liste déroulante et DataGridView sont synchronisés. La modification de la valeur de la zone de liste déroulante déplace le DataGridView au même enregistrement. Faire défiler le groupe de données dans la grille ou utiliser la barre d'outils de navigation permet de mettre à jour la zone de liste déroulante.

Le formulaire principal contient les contrôles zone de liste déroulante, contrôles bouton pour charger des données et deux contrôles label et une grille pour afficher la source de données et la valeur sélectionnée. **ArrayList**, **DataSet**et **DataView** vous permettent d'associer une valeur à chaque élément affiché dans le contrôle zone de liste déroulante. Par exemple, si l'utilisateur sélectionne l'entrée de produit `Chai` dans la table de produits liée à la zone de liste déroulante, Chai est affiché comme entrée sélectionnée, mais son `ProductId` associé est également disponible à travers la propriété SelectedValue. La propriété ValueMember vous permet de sélectionner l'élément qui contient la valeur associée. La propriété DisplayMember vous permet de sélectionner l'élément qui est affiché dans le contrôle zone de liste déroulante.

Création de cet exemple

La majeure partie de ce formulaire a été créée en y faisant glisser des composants, puis en utilisant des balises actives et des paramètres dans la fenêtre **Propriétés**. Les éléments suivants expliquent de façon succincte comment vous pourriez créer à partir de zéro la partie **DataGridView** du formulaire :

1. Créez un projet d'application Windows.
2. Avec `Form1` ouvert, sélectionnez la fenêtre de **Source de données**. Elle peut également être activée au moyen du menu **Données**.
3. Dans la fenêtre **Sources de données**, cliquez sur **Ajouter une nouvelle source de données**.
4. Dans l'**Assistant Configuration de source de données**, choisissez Base de données comme type de source de données.
5. Pour la connexion de données, choisissez un serveur qui contient Northwind.
6. L'étape suivante vous permet d'enregistrer la chaîne de connexion dans un fichier des paramètres de l'application fortement typé.
    a. Dans "Choisir vos objets de base de données", sélectionnez la table `Products`.
    b. Cliquez sur **Terminer** pour créer votre groupe de données typé pour la base de données Northwind. Vous pouvez consulter les résultats dans la fenêtre **Sources de données**.
7. De la fenêtre **Sources de données**, faites glisser la table `Products` vers `Form1`.
8. Au final, vous verrez un **DataGridView** lié aux données et des contrôles BindingNavigator ajoutés à la surface du Concepteur de formulaires. `NorthwindDataSet`, `ProductsTableAdapter`et `ProductsBindingSource` ajouté à la barre d'état des composants s'affichera également.

Chargement de données dans le formulaire

Dans cet exemple, vous chargez le formulaire avec les données sans aucun paramètre fourni par l'utilisateur. Le **Concepteur de DataSet** vous permet de tirer parti de DataAdapter réutilisables pour remplir `dsProducts2`.

Lorsque vous faites glisser la table **Employees** à partir de la fenêtre **Sources de données**, Visual Studio place automatiquement le code pour qu'il appelle la requête par défaut dans le TableAdapter de l'événement `Form.Load`. Dans cet exemple que ce code a été déplacé vers la méthode Click `btnDC` :

```
' Fill the Lookup Tables
Me.ProductsTableAdapter.Fill(Me.NorthwindDataSet.Products)
```

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples
Comment : lier un contrôle ComboBox ou ListBox Windows Forms aux données

Référence

ComboBox
DataSource
ArrayList

DataTable
DataView
Concepts
Sources de données prises en charge par les Windows Forms

Recherche de données, exemple

Download sample

Cet exemple montre comment afficher un sous-ensemble de données de votre base de données.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cet exemple requiert l'exemple de base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

La plus grande part de ce formulaire a été créée à l'aide des outils du concepteur dans Visual Basic. Voici un résumé expliquant comment vous pourriez créer ce formulaire en partant de zéro.

Pour créer cet exemple

1. Créez un nouveau projet **Application Windows**.

2. À l'aide de l'Assistant Configuration de source de données, créez une connexion à la base de données Northwind en sélectionnant la table `Customers` et enregistrez la chaîne de connexion dans le fichier de configuration de l'application.

3. Ouvrez la fenêtre Sources de données et faites glisser la table `Customers` sur le formulaire. Des contrôles DataGridView et BindingNavigator sont ajoutés au formulaire et configurés pour la table `Customers`.

4. Dans la balise active pour **DataGridView**, cliquez sur **Ajouter une requête**. Dans la boîte de dialogue, remplacez le **Nouveau nom de requête** par **FillByLastName**. Dans le **Texte de la requête**, ajoutez `WHERE LastName LIKE @lastName + '%'` à la fin de l'instruction `Select`. Un contrôle ToolStrip de recherche est ajouté au formulaire.

5. Double-cliquez sur **NorthwindDataSet** dans l'**Explorateur de solutions**. Le fichier de code NorthwindDataSet.xsd s'ouvre. Notez que la méthode `FillByLastName` est ajoutée à CustomersTableAdapter.

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples

Concepts

Nouveautés des données

Vue d'ensemble de TableAdapter

Validation des données, exemple

Download sample

Cet exemple illustre le **Concepteur de DataSet** et utilise les événements PropertyChanged pour valider une entrée d'utilisateur.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour exécuter cet exemple

1. Cliquez sur le bouton **Télécharger l'exemple** dans cette page. Dans la boîte de dialogue **Téléchargement de fichier**, cliquez sur **Ouvrir**.
2. Enregistrez les fichiers sur votre ordinateur.
3. Dans le menu **Fichier** de Visual Studio, pointez sur **Ouvrir**, puis cliquez sur **Projet/Solution**.
4. Recherchez le dossier où vous avez enregistré les fichiers et sélectionnez le fichier solution. Cela ouvre la solution dans Visual Studio.
5. Appuyez sur F5.

Configuration requise

Cet exemple requiert l'exemple de base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

Cet exemple illustre les fonctionnalités de validation des données disponibles dans les concepteurs Visual Studio. Deux règles arbitraires et fictives sont utilisées pour valider les données. Le code est ajouté à l'aide du **Concepteur de DataSet**. En double-cliquant sur un nom de colonne, vous créez l'événement qui modifie la colonne. En double-cliquant sur la liste de champs vous créez l'événement qui modifie la ligne. Dans ces gestionnaires d'événements, le code est ajouté pour valider les valeurs proposées :

```
Private Sub EmployeeDataTable_BirthDateChanging( _
    ByVal sender As EmployeeDataTable, _
    ByVal e As BirthDateChangeEventArg) Handles Me.BirthDateChanging

    Dim minimumAgeDate As New DateTime(DateTime.Now().Year - 18, _
        DateTime.Now().Month, DateTime.Now().Day)
    If e.ProposedValue > minimumAgeDate Then
        e.Row.SetColumnError(e.BirthDateColumn, _
            "Employees must be at least 18 years of age.")
    Else
        e.Row.SetColumnError(e.BirthDateColumn, "")
    End If
End Sub

Private Sub EmployeesDataTable_EmployeesRowChanging( _
    ByVal sender As System.Object, ByVal e As _
    EmployeesRowChangeEvent) Handles Me.EmployeesRowChanging
    If (e.Row.BirthDate = e.Row.HireDate) Then
        e.Row.RowError = e.Row.RowError & _
            "Birth date and hire date are the same."
    Else
        e.Row.ClearErrors()
    End If
End Sub
```

Lorsque des erreurs de ligne et de colonne sont définies, le contrôle DataGridView affiche une icône d'avertissement. Pour les erreurs de ligne, l'icône est affichée à gauche de la ligne. Pour les erreurs de colonne, l'icône est affichée dans la cellule.

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples

Référence

DataRow

Autres ressources

Validation des données

Création de base de données, exemple

Download sample

Cet exemple montre comment exécuter des instructions SQL à partir du code.

Les instructions SQL sont exécutées pour créer une base de données, une table, une procédure stockée et un affichage dans le code. Les instructions SQL sont ensuite exécutées pour compléter la table par les données de la base de données Northwind. Puis cette table est interrogée pour remplir un DataSet qui est, par la suite, lié à un DataGrid pour affichage.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5.

2. Si vous souhaitez de nouveau exécuter l'application après avoir créé la base de données, vous devrez fermer toutes les applications qui maintiennent une connexion active avec l'instance SQL Server qui contient la base de données de démonstration. Cela signifie que vous devrez fermer et rouvrir Visual Studio si vous avez ouvert une connexion à la base de données dans l'Explorateur de serveurs. D'autres applications, comme l'Analyseur de requêtes SQL, peuvent également maintenir une connexion ouverte. Par ailleurs, vous pouvez arrêter et redémarrer l'instance SQL Server à l'aide du Gestionnaire de services SQL Server dans votre barre d'état système.

Configuration requise

Cet exemple requiert la base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

L'application vous passe en revue les étapes de création et de remplissage de la base de données :

- **Créer la base de données** La classe SqlCommand utilise l'instruction CREATE DATABASE pour créer la base de données `How to Demo` .

- **Créer la table de données** La classe **SqlCommand** utilise l'instruction CREATE TABLE pour créer une table nommée `NW_Seafood`. La table inclut les champs de l'ID de produit, du nom de produit, de la quantité par unité et du prix unitaire.

- **Créer une procédure stockée** La classe **SqlCommand** utilise l'instruction INSERT INTO et l'instruction SELECT pour ajouter le code à la nouvelle table et récupérer les données de la table.

- **Créer une vue** La classe **SqlCommand** utilise l'instruction CREATE VIEW pour sélectionner des lignes dans la table

`NW_Seafood` .

- **Remplir la table** La classe **SqlCommand** utilise l'instruction EXECUTE pour exécuter une procédure qui récupère les lignes de la base de données Northwind et les ajoute à la table `NW_Seafood` .

- **Afficher les données dans la table** La classe **SqlCommand** utilise l'instruction SELECT pour remplir un **DataSet** qui est ensuite utilisé comme DataSource d'un contrôle **DataGrid**. Les objets de style de la table et des colonnes sont utilisés pour personnaliser l'apparence du **DataGrid**.

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples

Référence

SqlCommand Class

Concepts

Comparaison entre ADO.NET et ADO

Autres ressources

Vue d'ensemble d'ADO.NET

Données locales, exemple

Download sample

Cet exemple présente les fonctionnalités de données locales de Visual Studio 2005, notamment la fenêtre **Sources de données**, TableAdapters et le composant BindingSource. Cet exemple montre également comment vous pouvez générer des formulaires liés aux données sans devoir écrire de code, et comment se connecter à un fichier de base de données de votre projet.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

   Installez ensuite la base de données Northwind.

Pour exécuter l'exemple

1. Naviguez jusqu'à la page du Centre de téléchargement Microsoft (http://www.microsoft.com/downloads).

2. Dans la zone **Product/Technology** du volet **Search for a Download**, sélectionnez **Access**.

3. Dans la zone **Keywords**, tapez **Northwind**.

4. Cliquez sur **Go**.

5. Dans la page des résultats, suivez le lien vers **Access 2000 Tutorial: Northwind Traders Sample Database**, puis les instructions de téléchargement et d'installation.

6. Faites glisser le fichier **Nwind.mdb** dans l'**Explorateur de solutions** (sur le nœud de projet **Données locales**).

   L'**Assistant Configuration de source de données** s'ouvre.

7. Cliquez sur **Annuler** pour quitter l'Assistant.

8. Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Choisissez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Configuration requise

Cet exemple nécessite l'exemple de base de données Access nommé Nwind.mdb. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

Visual Studio 2005 introduit la fenêtre **Sources de données**, laquelle vous permet de définir des liaisons non seulement avec des données serveur, mais également avec des données locales, des services Web et des objets métier. En faisant glisser une table jusqu'à un formulaire, vous obtenez un formulaire lié aux données, ce qui gère la plupart des étapes que vous deviez effectuer manuellement dans les versions antérieures.

Les étapes de base pour créer une application comme celle-ci sont les suivantes :

1. Créez un projet **Application Windows**.
2. Ajoutez une nouvelle source de données.
3. Sélectionnez les contrôles d'affichage pour les colonnes de la table ou acceptez les sélections par défaut.
4. Faites glisser la table vers votre formulaire.
5. Ajoutez le code d'insertion, de mise à jour et de suppression des enregistrements. Examinez le code de l'événement `Click` de l'élément **Enregistrer** sur ToolStrip.

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples

Autres ressources

Exemples de données

Liaison d'objets, exemple

Download sample

Cet exemple montre comment lier des contrôles aux objets autres que des composants de base de données.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

La solution de l'exemple contient deux projets. Le projet `CustomerLibrary` définit plusieurs classes métier, notamment `Customers`, `Orders` et `CustomerManager`. La classe `CustomerManager` crée des instances de ces classes. Ces instances sont affichées dans le projet client.

Dans le **Concepteur de formulaires**, deux contrôles BindingSource qui lient les objets métier aux classes métier sont créés. Les contrôles situés sur le formulaire sont ensuite liés aux contrôles **BindingSource**. Pour plus d'informations, consultez Comment : établir une connexion à des données dans un objet.

Voir aussi

Tâches

Comment : établir une connexion à des données dans un objet

Sérialisation, exemple

Download sample

Cette solution montre comment sérialiser et désérialiser un objet.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cette application contient deux classes : l'une avec sérialisation standard, l'autre avec sérialisation personnalisée. L'interface utilisateur inclut des boutons permettant de sérialiser et désérialiser les deux classes. La classe avec sérialisation standard peut être sérialisée à un fichier SOAP ou un fichier binaire. La classe avec sérialisation personnalisée est sérialisée à un fichier SOAP. Un contrôle TextBox est utilisé pour afficher les fichiers SOAP. Les deux classes ont l'attribut SerializableAttribute. Les classes sont décrites dans le tableau suivant.

Membre de classe	**SerializationClass**	**CustomSerializationClass**
`PublicVariable`	La variable publique est sérialisée et désérialisée sans modification.	La variable publique est sérialisée et désérialisée sans modification.
`PublicProperty`	La variable privée qui prend en charge la propriété est sérialisée et désérialisée sans modification.	La variable privée qui prend en charge la propriété est sérialisée comme -1, indépendamment du jeu de valeur de l'interface utilisateur. Par conséquent, elle est désérialisée comme -1.
`NonSerializedVariable`	Cette variable n'est pas sérialisée, car elle est marquée avec NonSerializedAttribute. La variable publique n'est pas sérialisée et la valeur désérialisée est 0.	Bien que la variable soit marquée par **NonSerializedAttribute**, elle est sérialisée, car la sérialisation personnalisée substitue ce paramètre. La variable publique est sérialisée et désérialisée sans modification.

Trois types de sérialisation sont illustrés :

- **Format SOAP** `Class1` est sérialisé à l'aide de la classe SoapFormatter.
- **Format binaire** `Class1` est sérialisé à l'aide de classe BinaryFormatter.

- **Format personnalisé** `Class2` implémente l'interface ISerializable, y compris la méthode GetObjectData, pour effectuer une sérialisation personnalisée.

Voir aussi

Référence

System.Runtime.Serialization.Formatters.Soap
Serialize
Deserialize
ISerializable
GetObjectData
SerializableAttribute

Procédure stockée, exemple

Download sample

Cette application montre comment créer et utiliser par programme des procédures stockées avec ADO.NET et Microsoft SQL Server.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cet exemple requiert l'exemple de base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

Les tâches illustrées sont les suivantes :

- Créer une nouvelle base de données à l'aide de commandes SQL et de fichiers de ressources pour contrôler les scripts à exécuter. Une nouvelle base de données est créée avec une nouvelle table. La table est ensuite remplie à l'aide de scripts supplémentaires. Un script supplémentaire est exécuté pour créer les procédures stockées utilisées dans cet exemple.
- Exécuter une procédure stockée qui ne requiert aucun paramètre à l'aide d'un objet SqlDataReader.
- Exécuter une procédure stockée qui requiert un paramètre d'entrée et remplir un contrôle Grid avec les résultats d'un SqlDataAdapter.
- Exécuter une procédure stockée qui requiert à la fois des paramètres d'entrée et des paramètres de sortie, et qui utilise une valeur de retour.

Voir aussi

Tâches

Comment : créer et exécuter une instruction SQL qui retourne une seule valeur
Comment : exécuter une procédure stockée qui ne retourne aucune valeur
Comment : installer et dépanner des composants de base de données pour les exemples

Concepts

Paramètres des commandes d'adaptateur de données
Extraction de données à l'aide d'un DataReader

Vue d'ensemble de TableAdapter
Nouveautés des données

Classe XMLDocument, exemple

Download sample

Cet exemple montre comment utiliser la classe XmlDocument qui fait partie de l'espace de noms System.Xml. Cette classe expose le Document Object Model XML ou DOM qui peut être utilisé pour manipuler des données XML.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le formulaire est divisé en trois sections :

- Une liste de tâches.
- Une zone d'affichage pour le XML.
- Une zone d'affichage pour les résultats de l'exécution.

Les trois zones sont chargées et mises à jour dynamiquement. L'utilisation d'une collection d'éléments au lieu d'une série de contrôles bouton vous permet d'ajouter autant d'options que vous le souhaitez, sans avoir à réordonner les contrôles sur le concepteur.

Lorsque l'exemple charge, il vérifie l'existence de trois fichiers dans le dossier \bin du dossier du projet :

- Simple.xml
- New.xml
- Bad.xml

Si ces fichiers existent, vous serez en mesure faire votre choix parmi un ensemble de commandes chargé dans un contrôle CheckedListBox. Vous aurez besoin de l'autorisation de lecture/écriture dans le dossier \bin, étant donné que certaines de ces commandes requièrent que les modifications soient enregistrées sur le disque.

Voir aussi

Tâches

Comment : installer et dépanner des composants de base de données pour les exemples

Référence
XmlDocument
System.Xml

Exemples de Windows Forms Visual Basic

Ces exemples illustrent les tâches courantes des applications Windows Forms.

Dans cette section

Animation, exemple

Inclut le déplacement de texte, une balle en mouvement et une image animée.

Presse-papiers, exemple

Présente l'objet **My.Clipboard** pour copier et coller du texte, des entiers, des images et des objets.

Contrôle utilisateur de dessin personnalisé, exemple

Montre comment créer et utiliser des contrôles utilisateur dessinés de manière personnalisée.

Composants Dialog, exemple

Illustre la sélection des fichiers, des couleurs et des dossiers.

Glisser-déplacer, exemple

Illustre la fonctionnalité de glisser-déplacer sur du texte, des images et des nœuds d'arborescence.

Création de contrôle dynamique, exemple

Ajoutez des contrôles au moment de l'exécution à l'aide de fichiers d'entrée XML.

Application de type Explorer, exemple

Contient deux formulaires qui possèdent des fenêtres à volet fractionné, semblable à **Explorateur Windows**.

Pinceaux GDI+, exemple

Afficher les effets des options de pinceau (brush en anglais).

Images GDI+, exemple

Manipule des images, y compris en effectuant des zooms et des découpages.

Stylet GDI+, exemple

Montre les effets des options de stylet.

Texte GDI+, exemple

Montre les effets des options de texte.

Fichiers d'aide, exemple

Montre l'aide compilée, l'aide HTML et les info-bulles.

Zone de liste et zone de liste déroulante, exemple

Illustre plusieurs tâches de liste courantes.

Menus, exemple

Crée et fusionne des menus par programme.

NotifyIcon, exemple

Crée une application accessible depuis la barre d'état système.

Impression, exemple

Utilise l'objet **My.Printers**.

Expressions régulières, exemple

Valide l'entrée d'utilisateur à l'aide de Regex.

Contrôle StatusStrip, exemple

Illustre les volets StatusBar de texte et dessinés par utilisateur.

Validation de texte, exemple

Valide le texte à l'aide du MaskedTextBox.

Mise à jour de l'interface utilisateur à partir d'un thread, exemple

Montre comment mettre à jour l'interface utilisateur à partir d'un thread séparé.

Formulaires de niveau supérieur, exemple

Contrôle l'apparence des applications dans la **Barre des tâches**.

Héritage visuel, exemple

Crée de nouveaux formulaires en héritant de formulaires existants.

Navigateur Web, exemple

Crée un navigateur Web.

Référence

Applications Windows Forms dans le .NET Framework

Informations sur la conception des applications Windows Forms.

Contrôles Windows Forms

Informations sur les contrôles Windows Forms.

Rubriques connexes

Exemples de données

Ces exemples illustrent des tâches de manipulation de données.

Exemples de langages Visual Basic

Ces exemples illustrent des concepts du langage Visual Basic.

Exemples de composants serveur Visual Basic

Ces exemples illustrent des applications qui interagissent avec les composants du système d'exploitation Windows.

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Voir aussi

Tâches

Contrôle FlowLayoutPanel, exemple
Contrôle MaskedTextBox, exemple
Contrôle TableLayoutPanel, exemple
Contrôle BindingNavigator, exemple
Contrôle SplitContainer, exemple

Autres ressources

Contrôles Windows Forms, exemples

Animation, exemple

Download sample

Cet exemple montre comment implémenter une animation avec les objets de l'espace de noms System.Drawing.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple utilise un objet Timer pour contrôler les modifications de l'animation et les contrôles RadioButton pour sélectionner l'objet de l'animation. Trois styles d'animation sont affichés :

- **Animation de frames.** L'œil faisant un clin d'œil est implémenté comme une série d'images statiques (frames). À chaque graduation de minuterie, le frame suivant est affiché.

- **Objets volants.** La taille et la vitesse de la balle en mouvement sont relatives à la zone cliente. Cette animation est implémentée en dessinant dans l'objet Graphics du formulaire.

- **Animation de texte.** Le texte est dessiné à l'aide de l'objet LinearGradientBrush. Un pinceau différent est créé à chaque graduation de minuterie en modifiant les paramètres en **LinearGradientBrush**. Les paramètres déplacent les points de départ et de fin du dégradé.

Voir aussi

Référence

Graphics

LinearGradientBrush

Concepts

Vue d'ensemble des graphismes

Presse-papiers, exemple

Download sample

Cet exemple montre comment placer et récupérer des éléments du Presse-papiers dans plusieurs formats.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Cliquez sur **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

Les commandes de menu du formulaire principal contrôlent les deux tâches principales d'utilisation du Presse-papiers, c'est-à-dire la copie dans le Presse-papiers et la récupération à partir du Presse-papiers. Six types de données sont copiés et collés à l'aide de l'objet My.Computer.Clipboard :

- **Image** Cette fonctionnalité utilise la méthode My.Computer.Clipboard.SetImage et la méthode My.Computer.Clipboard.GetImage pour copier l'image d'un contrôle PictureBox vers un autre. Le fichier image est stocké dans **My.Resources**.

- **Texte** Cette fonctionnalité utilise la méthode My.Computer.Clipboard.SetText Method et la méthode My.Computer.Clipboard.GetText pour copier le texte d'un contrôle TextBox vers un autre. La méthode My.Computer.Clipboard.SetText est surchargée. La surcharge utilisée dans cette tâche n'a aucun paramètre et elle copie tout type de texte qui se trouve dans le Presse-papiers.

- **Texte enrichi** Comme la fonctionnalité de "texte", cette fonctionnalité utilise la méthode My.Computer.Clipboard.SetText et la méthode My.Computer.Clipboard.GetText pour copier et coller le texte. Cette fonctionnalité utilise la surcharge de la méthode My.Computer.Clipboard.SetText qui accepte une énumération TextDataFormat. La valeur passée est Rtf, par conséquent, seul le texte enrichi est copié.

- **HTML** Comme la fonctionnalité "texte enrichi", celle-ci utilise une valeur (Html) de l'énumération **TextDataFormat** pour récupérer uniquement les éléments au format HTML du Presse-papiers.

- **Instance de classe** La classe `Pixel` est définie dans le projet et possède l'attribut `Serializable`. La méthode My.Computer.Clipboard.SetData et la méthode My.Computer.Clipboard.GetData peuvent être utilisées pour copier et coller les données. La méthode `LoadPixel` crée une chaîne de texte à écrire dans le formulaire.

- **Fichier** Un fichier est copié dans le Presse-papiers à partir de l'**Explorateur Windows**. Le nom de fichier est récupéré à l'aide de la méthode My.Computer.Clipboard.GetFileDropList.

Dans chaque cas, les données sont copiées depuis le contrôle de code source de la colonne de gauche et collées dans le contrôle de destination de la colonne de droite.

Voir aussi

Tâches

Comment : écrire dans le Presse-papiers en Visual Basic
Comment : effacer le Presse-papiers en Visual Basic
Comment : lire le Presse-papiers en Visual Basic
Comment : récupérer une image du Presse-papiers en Visual Basic

Référence

My.Computer.Clipboard, objet

Contrôle utilisateur de dessin personnalisé, exemple

Download sample

Cet exemple montre comment créer et utiliser des contrôles utilisateur dessinés de manière personnalisée.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'affiche.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple montre comment créer des contrôles utilisateur dessinés de manière personnalisée, les utiliser sur un formulaire, inviter le contrôle à répondre aux événements et définir des propriétés pour une utilisation dans le concepteur. Les contrôles qui sont créés sont des contrôles qui conservent le score. L'un est un scoreboard numérique qui affiche un nombre dans un format numérique. L'autre est un boulier sur lequel on fait glisser des boules d'un bout à l'autre pour suivre un score. Les deux contrôles répondent à divers événements et possèdent de nombreuses propriétés disponibles au moment du design.

- **Dessin personnalisé** Les deux contrôles personnalisés de cet exemple montrent comment hériter de la classe UserControl et substituer OnPaint. Un contrôle utilisateur dessiné de manière personnalisée hérite de **UserControl**. Il est possible d'effectuer un dessin personnalisé en substituant la méthode **OnPaint** du contrôle. La méthode **OnPaint** fournit l'argument PaintEventArgs à partir duquel le rectangle de coupure (Rectangle) et l'objet Graphics (Graphics) sont obtenus.

- **Événements de contrôle personnalisé** Les contrôles personnalisés peuvent répondre aux événements auxquels un contrôle normal pourrait répondre. Pour ce faire, il substitue la méthode du gestionnaire d'événements dans la classe **UserControl**. Les deux contrôles personnalisés de cet exemple montrent comment répondre à divers événements tels que MouseUp et Click.

- **Propriétés** Il est souvent préférable de créer des propriétés pour un contrôle qui peut être modifié au moment du design, à l'aide de Visual Studio. Les deux contrôles personnalisés de cet exemple montrent comment créer des propriétés **Public** qui apparaîtront dans la fenêtre **Propriétés** de Visual Studio.

Voir aussi

Tâches

Dessin d'une image sur votre formulaire

Rendre les graphiques apparents

Autres ressources

Peinture et rendu personnalisés des contrôles

Composants Dialog, exemple

Download sample

Cet exemple illustre les composants OpenFileDialog, SaveFileDialog, ColorDialog et FontDialog.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple illustre les composants **OpenFileDialog**, **SaveFileDialog**, **ColorDialog** et **FontDialog**. L'exemple de formulaire contient un contrôle TabControl avec deux pages. La première page d'onglets utilise les quatre composants pour ouvrir un fichier, enregistrer un fichier et définir la police et couleur du texte. La deuxième page d'onglets utilise les propriétés FileNames et Multiselect pour autoriser la sélection de plusieurs fichiers.

Voir aussi

Référence

FileNames
Multiselect

Autres ressources

OpenFileDialog, composant (Windows Forms)
SaveFileDialog, composant (Windows Forms)
ColorDialog, composant (Windows Forms)
FontDialog, composant (Windows Forms)

Glisser-déplacer, exemple

Download sample

Cet exemple comporte trois exemples de fonctionnalités de glisser-déplacer sur un formulaire dans une application Windows :

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple est composé de trois exemples de glisser-déplacer.

- **TextBox** Montre comment empêcher un déplacement sur un contrôle. Vous pouvez faire glisser du texte d'un contrôle TextBox vers les deux autres, l'un des deux n'ayant pas la valeur **True** définie pour la propriété AllowDrop.

- **TreeView** Montre comment déplacer des nœuds d'un contrôle TreeView à un autre.

- **PictureBox** Montre comment copier des images. Vous pouvez faire glisser des images d'un PictureBox vers un autre. La propriété AllowDrop n'apparaît pas dans la fenêtre **Propriétés**, et a donc la valeur **True** dans le code.

Les données à faire glisser doivent être une instance de la classe `String`, `Bitmap` ou `MetaFile`, ou un objet qui implémente l'interface ISerializable ou IDataObject.

Voir aussi

Tâches

Procédure pas à pas : exécution d'opérations de glisser-déplacer dans Windows Forms

Référence

Metafile Class
ISerializable Interface
IDataObject Interface
AllowDrop

Création de contrôle dynamique, exemple

Download sample

Cet exemple montre comment ajouter des contrôles à un formulaire dans une application Windows au moment de l'exécution.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Plusieurs options permettant d'ajouter des contrôles à des formulaires sont illustrées. Les contrôles sont ajoutés au formulaire principal en déclarant une nouvelle instance d'un contrôle et en ajoutant la référence à la collection ControlCollection du formulaire. Un scénario plus compliqué illustre la création d'une instance d'un nouveau formulaire et l'ajout de contrôles au nouveau formulaire. Les contrôles à créer sont spécifiés dans un fichier XML.

Voir aussi

Référence

System.Windows.Forms

Control

Autres ressources

Création de gestionnaires d'événements dans les Windows Forms

Gestion et déclenchement d'événements

Application de type Explorer, exemple

Download sample

Cet exemple contient deux formulaires qui fonctionnent comme l'Explorateur : `DirectoryScanner` et `ExplorerStyleViewer`.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple contient deux formulaires avec une interface de style Explorateur : un scanneur de répertoire et une visionneuse de style Explorateur. Les fichiers qui prennent en charge les deux formulaires sont contenus dans des dossiers séparés du projet.

- **DirectoryScanner** Il s'agit d'une application simple qui analyse tous les répertoires et sous-répertoires situés dans tous les lecteurs logiques ou dans un répertoire de démarrage sélectionné par l'utilisateur. La liste de lecteurs est obtenue à l'aide de la méthode Directory.GetLogicalDrives. Un contrôle Treeview affiche la structure de répertoires qui reflète la numérisation la plus récente. Les répertoires apparaissent en vert, en jaune ou en rouge en fonction de leur taille totale, sous-répertoires et fichiers inclus. La méthode Directory.GetFiles est utilisée pour récupérer la liste des fichiers et la classe FileInfo est utilisée pour obtenir la taille de fichier.

- **ExplorerStyleViewer** Il s'agit d'une version plus simple de l'application de l'**Explorateur Windows**. `ExplorerStyleViewer` affiche plus d'informations de fichier que `DirectoryScanner` en utilisant la propriété FileSystemInfo.Attributes. Il montre comment associer des icônes aux types de fichier à l'aide de la propriété TreeView.ImageList. Il permet à l'utilisateur d'exécuter une application associée au type de fichier (si une telle association existe) en double-cliquant sur le fichier (comme dans l'**Explorateur Windows**). Les applications sont démarrées à l'aide de la méthode Process.Start.

Voir aussi

Référence

Directory Class
FileInfo Class
TreeView.ImageList Property
ListView Class
TreeView Class

Autres ressources

ListView, contrôle (Windows Forms)
TreeView, contrôle (Windows Forms)

Pinceaux GDI+, exemple

Download sample

Cet exemple illustre diverses fonctionnalités de la classe Brush.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

L'utilisateur peut sélectionner plusieurs paramètres différents qui influencent l'affichage du pinceau. Après chaque sélection de paramètre, les formes sélectionnées par utilisateur sont redessinées dans une zone d'image illustrant le nouveau pinceau. Certains d'objets plus avancés tels que PathGradientBrush et LinearGradientBrush sont également inclus.

Voir aussi

Tâches

Comment : créer des objets graphiques pour le dessin

Autres ressources

À propos du code managé GDI+

Images GDI+, exemple

Download sample

Cet exemple d'application montre comment manipuler des images à l'aide de GDI+.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Les idées présentées dans cet exemple sont les suivantes :

- **Zoom** La fonctionnalité de zoom modifie la taille d'une image à l'écran. L'importance du zoom effectué est généralement exprimée en pourcentage de la taille d'origine de l'image. Effectuer un zoom ne modifie pas la taille de l'image d'origine, mais uniquement la taille de l'image qui s'affiche à l'écran. Cela est illustré dans la méthode `Zoom` de cet exemple.

- **Redimensionner** La fonctionnalité de redimension modifie la taille réelle de l'image. La largeur et la hauteur sont toutes deux modifiées selon un rapport de redimension. Cela est illustré dans la méthode `ResizeImage` de cet exemple.

- **Retourner et faire pivoter** Le retournement reflète une image par rapport à l'axe X ou l'axe Y. Le résultat constitue une image miroir ou une image renversée. La rotation fait pivoter l'image d'un certain nombre de degrés à partir de son centre. Pour les images, la rotation s'effectue très souvent selon un multiple de 90 degrés. Le retournement et la rotation sont illustrés dans la méthode `RotateFlip` de cet exemple.

- **Négatif et nuances de gris** La création d'une image négative ou à nuances de gris s'effectue à l'aide d'un ColorMatrix. Appliqué à une image, **ColorMatrix** est la base d'une transformation de couleur. Cela est illustré dans les méthodes `DrawNegativeImage` et `ConverttoGrayScale` de cet exemple.

- **Rogner** Le rognage prélève une région d'une image et crée une image séparée à partir de ce prélèvement. Cela est illustré dans la méthode `CropButton_Click` de cet exemple.

- **Miniatures** Les images peuvent être enregistrées à leur taille normale ou en tant que miniatures. Une façon de créer une miniature consiste à créer un clone de l'image, à le redimensionner et à l'enregistrer en tant que miniature. Cela est illustré dans la méthode `SaveThumbnailAs_Click` de cet exemple.

Voir aussi

Tâches

Comment : créer des objets graphiques pour le dessin
Comment : rogner et mettre des images à l'échelle
Comment : faire pivoter, refléter et incliner des images
Comment : créer des images miniatures

Autres ressources

Graphiques et dessins dans les Windows Forms

Stylet GDI+, exemple

Download sample

Cet exemple illustre diverses fonctionnalités de la classe Pen.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Illustration

Plusieurs contrôles permettent à l'utilisateur de sélectionner les caractéristiques du stylet :

- Width
- DashStyle
- MiterLimit
- StartCap
- EndCap
- DashCap
- LineJoin
- Alignment

L'utilisateur peut également choisir entre trois transformations :

- ScaleTransform
- RotateTransform
- TranslateTransform

Une fois que **Pen** et les transformations sont configurées, l'utilisateur peut sélectionner l'illustration à dessiner :

- **Lignes** Les lignes sont dessinées à l'aide de la méthode DrawLine.

- **Lignes qui se croisent** Les lignes sont dessinées à l'aide de la méthode DrawLines.
- **Cercles et courbes** Les lignes sont dessinées à l'aide des méthodes DrawEllipse et DrawArc.

Voir aussi

Référence

Pen

Autres ressources

À propos du code managé GDI+

Texte GDI+, exemple

Download sample

Cet exemple illustre certaines des nombreuses fonctionnalités disponibles lors de l'utilisation de GDI+ pour travailler avec du texte. Plusieurs effets sont décrits, y compris l'ombre, le relief, le normal centré, l'inclinaison et la réflexion.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Sélectionnez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

Tandis que certains effets peuvent être implémentés en définissant les propriétés des classes Brush et Pen, d'autres effets nécessitent du code :

- **Ombre** Pour créer l'ombre, le texte est dessiné deux fois. La première fois, il est en gris et décalé. La deuxième fois, il est en noir.

- **Relief** Pour créer l'effet de relief, le texte est dessiné deux fois. La première fois, il est en noir et décalé. La deuxième fois, il est dessiné en blanc, la couleur actuelle de l'arrière-plan.

- **Normal centré** Pour obtenir cet effet, le texte est dessiné à plusieurs reprises en noir, en déplaçant le texte vers le haut et à droite à chaque fois. Il est ensuite dessiné dans la couleur principale.

- **Inclinaison** Dans le contexte du dessin, inclinaison signifie incliner une image. Le texte est incliné par l'appel à la méthode Shear de la propriété Transform de l'objet Graphics.

- **Réflexion** La réflexion du texte requiert la mesure de la hauteur du texte, la mise à l'échelle du texte et le repositionnement de l'origine. La hauteur retournée par la méthode MeasureString comprend un espace supplémentaire

pour les hampes inférieures et l'espace blanc, mais le texte sera reflété autour de la ligne de base, laquelle est la ligne sur laquelle se trouvent toutes les majuscules. Pour calculer la hauteur au-dessus de la ligne de base, utilisez la méthode GetCellAscent. Étant donné que **GetCellAscent** retourne une valeur métrique de conception, celle-ci doit être convertie en pixels et mise à l'échelle en fonction de la taille de la police. Le texte est réfléchi à l'aide d'une transformation de mise à l'échelle, mais une nouvelle origine est d'abord définie pour empêcher le texte d'être dessiné à l'extérieur de la zone visible du formulaire. Enfin, le texte refléchi est dessiné en premier, présentant la classe GraphicsState. Le texte réfléchi est dessiné en premier uniquement pour présenter l'utilisation de l'objet **GraphicsState**. Enfin, le texte droit est dessiné.

Voir aussi

Référence

Graphics Class
Graphics.DrawString Method
Pen Class
Graphics.Transform Property
Graphics.Save Method
Brush
Shear
MeasureString
GraphicsState

Fichiers d'aide, exemple

Download sample

Cet exemple indique comment ajouter de l'aide dans une application Windows à l'aide des contrôles ToolTip, HelpProvider et ErrorProvider.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'affiche.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Chaque page avec onglet du TabControl affiche un type d'aide différent :

- **Info-bulle** Un composant **ToolTip** est ajouté au formulaire. Cela ajoute la propriété `ToolTip1` dans chaque contrôle du formulaire.

- **PopUp** Un composant **HelpProvider** nommé `hpAdvancedCHM` est ajouté au formulaire. Cela ajoute la propriété **Help String on hpAdvancedCHM** à chaque contrôle défini avec une chaîne d'aide. La propriété **ShowHelp on hpAdvancedCHM** est également ajoutée à chaque contrôle et prend la valeur **True**. La propriété HelpButton du formulaire se voit affecter la valeur **True** pour afficher le bouton contenant un point d'interrogation dans l'angle situé à proximité du bouton Fermer. Les boutons de réduction et d'agrandissement doivent être désactivés pour que le bouton d'aide apparaisse.

- **Aide HTML** Un composant **HelpProvider** nommé `hpAdvancedCHM` est ajouté au formulaire. Cela ajoute les propriétés **HelpKeyword on hpAdvancedCHM**, **HelpNavigator on hpAdvancedCHM** et **ShowHelp on hpAdvancedCHM** à chaque contrôle. La propriété **ShowHelp on hpAdvancedCHM** a la valeur **True**. Le fichier d'aide compilé est ouvert à la page d'aide du mot clé spécifié. La propriété **HelpButton** du formulaire se voit affecter la valeur **True** pour afficher le bouton contenant un point d'interrogation dans l'angle situé à proximité du bouton Fermer. Les boutons de réduction et d'agrandissement doivent être désactivés pour que le bouton d'aide apparaisse.

- **Fournisseur d'erreur** Le composant **ErrorProvider** est utilisé pour montrer comment donner à l'utilisateur un avertissement visible sur un contrôle qui ne valide pas le type de données correct.

Voir aussi

Référence

Form.HelpButton Property

Autres ressources

HelpProvider, composant (Windows Forms)
ToolTip, composant (Windows Forms)
ErrorProvider, composant (Windows Forms)

Zone de liste et zone de liste déroulante, exemple

Download sample

Cet exemple montre plusieurs tâches ListBox et ComboBox de base, telles que l'ajout d'éléments, la création d'une liaison à une table de base de données et l'accès à l'élément sélectionné.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cet exemple requiert la base de données Northwind. Pour plus d'informations, consultez Comment : installer et dépanner des composants de base de données pour les exemples.

Démonstrations

Les pages avec onglet du contrôle onglet illustrent différentes techniques :

- **Add Items** L'exemple récupère un tableau des objets Process représentant les processus en cours d'exécution sur l'ordinateur. Les objets Process sont ajoutés à la collection Items des contrôles **ListBox**. La propriété DisplayMember du contrôle **ListBox** est utilisée pour spécifier quelle propriété de l'objet **Process** s'affiche dans le contrôle **ListBox**.

- **Bind to DataTable** Cette page d'onglets contient du code permettant de remplir un DataTable avec une liste de tous les fichiers de votre dossier **Mes documents**, puis de lier la zone de liste à ce **DataTable**.

- **Bind to Array** Cette page d'onglets contient du code semblable à celui de la page **Add Items**. Au lieu d'ajouter les objets **Process** un par un dans la collection **Items**, l'ensemble du tableau d'objets **Process** est utilisé comme DataSource du contrôle **ListBox**. Les propriétés ValueMember et **DisplayMember** du contrôle **ListBox** sont utilisées pour contrôler l'affichage et la récupération des éléments.

- **ComboBox** Cette page d'onglets contient du code permettant de lier un contrôle **ComboBox** à un **DataTable** récupéré de SQL Server à l'aide du nouveau TableAdapter et des fonctionnalités de groupe de données typé. Cet exemple s'attend à trouver l'exemple de base de données Northwind dans SQL Server sur l'ordinateur local. La page contient des contrôles vous permettant de spécifier certaines des propriétés de présentation qui affectent le comportement du contrôle **ComboBox**.

Voir aussi

Référence

ListControl.DisplayMember Property
ListControl.ValueMember Property
ListBox.SelectedIndexChanged Event
ListBox.SelectedIndices Property
ListBox.SelectedItems Property
ListBox.SelectionMode Property
Process Class
Process Class

Concepts

Nouveautés des données

Autres ressources

ListBox, contrôle (Windows Forms)
ComboBox, contrôle (Windows Forms)

Menus, exemple

Download sample

Cet exemple présente l'utilisation de contrôles MenuStrip dans les applications Windows.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple vous permet de tester le contrôle **MenuStrip** dans une application Windows simple. L'exemple illustre plusieurs fonctionnalités des bandes de menus : les contrôles, les événements de menu, les propriétés de menu, la gestion de plusieurs événements, l'assignation de gestionnaires d'événements à des événements de menu, l'ajout et la suppression d'objets dans la bande de menus au moment de l'exécution, l'affichage de raccourcis et l'utilisation d'un StatusStrip comme indicateur. La liste suivante détaille plusieurs fonctionnalités.

- **Raccourcis** Certains éléments de menu incluent des raccourcis, par exemple le menu **Options List** sur `Form1`. La propriété ShowShortcutKeys affiche ou masque cette information sur les menus. L'utilisation de touches de raccourci améliore l'accessibilité d'une application.

- **Contrôle StatusStrip** L'élément **Status Strip** dans le menu **View** utilise un élément de menu activé pour basculer la visibilité du contrôle **StatusStrip**.

- **Liste de cases à cocher** Le menu **Checked List** sur `MainMenuStrip` montre comment créer une collection mutuellement exclusive d'éléments de menu activés. Il s'agit d'une collection où deux éléments de l'ensemble ne peuvent être activés simultanément. Pour obtenir un exemple de cette opération, consultez le gestionnaire d'événements `MenuOption_Click`.

- **Ajout des éléments de menu au moment de l'exécution** La procédure `CreateInitialMenus` montre comment créer et ajouter un ToolStripMenuItem, ToolStripSeparator, ToolStripComboBox, ToolStripButton, ToolStripSplitButton ou ToolStripLabel au moment de l'exécution. Utilisez la propriété DropDownItems pour ajouter ou supprimer des éléments de la collection.

- **AddHandler** La procédure `CreateInitialMenus` utilise **AddHandler** pour assigner des gestionnaires d'événements à des événements spécifiques pour les éléments de menu qui sont créés pendant l'exécution.

Voir aussi

Tâches

Comment : créer un ToolStrip Windows Forms de base avec des éléments standard à l'aide du concepteur

Référence

Vue d'ensemble du contrôle ToolStrip (Windows Forms)

Concepts

Résumé de la technologie ToolStrip

Autres ressources

MenuStrip, contrôle (Windows Forms)

ToolStrip, contrôle (Windows Forms)

StatusStrip, contrôle

NotifyIcon, exemple

Download sample

Cet exemple montre comment créer un programme qui affiche une icône dans la zone de notification d'état de la barre des tâches.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'affiche.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cette application utilise le contrôle NotifyIcon et les ContextMenu qui lui sont associés pour accéder et afficher les données système :

- Le temps, en minutes, écoulé depuis le dernier redémarrage de l'ordinateur de l'utilisateur
- Le fuseau horaire actuel de l'utilisateur
- La date actuelle
- La version du .NET Framework qui est installée sur l'ordinateur de l'utilisateur
- La version actuelle du système d'exploitation sur lequel l'utilisateur exécute l'application

Voir aussi

Référence

NotifyIcon

Autres ressources

NotifyIcon, composant (Windows Forms)

Impression, exemple

Download sample

Cet exemple d'application montre comment imprimer le texte et configurer l'imprimante.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Sélectionnez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

L'exemple imprime quelques paragraphes de texte, en montrant l'aperçu avant impression, la configuration de l'impression et la mise en page.

Voir aussi

Autres ressources

Visite guidée de Visual Basic

Expressions régulières, exemple

Download sample

Cet exemple d'application montre comment utiliser des expressions régulières. Le formulaire principal contient un TabControl avec trois pages d'onglets.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Une connexion à Internet est requise pour la fonctionnalité de capture d'écran de l'exemple.

Démonstrations

Cet exemple d'application montre comment utiliser des expressions régulières. Les pages d'onglets présentent les informations suivantes :

- **Techniques d'analyse de base** Cette page d'onglets utilise la classe Regex pour rechercher des nombres dans une chaîne, ainsi que des codes postaux valides, des adresses de messagerie et des dates.

- **Capture d'écran** Cet onglet montre une utilisation plus avancée des expressions régulières. Vous pouvez entrer une adresse Web et afficher les liens ou images qu'elle contient. Les attributs de balise HTML sont affichés dans un contrôle ListView. Vous pouvez double-cliquer sur une image pour l'afficher ou double-cliquer sur un lien pour ouvrir Internet Explorer et afficher la page.

- **Test** Cet onglet offre un moyen commode d'expérimenter les expressions régulières. Il montre également comment effectuer une recherche et un remplacement ainsi qu'un fractionnement de chaîne.

Voir aussi

Référence

Regex Class

Autres ressources

Expressions régulières du .NET Framework

Contrôle StatusStrip, exemple

Download sample

Cet exemple montre comment représenter visuellement différents types de données dans le contrôle StatusStrip.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Plusieurs types de données sont affichés :

- **Texte** Le texte est affiché dans un volet.
- **État du clavier** L'état des touches VERR.MAJ et VERR.NUM est affiché sous forme de texte dans un volet.
- **Date** La date est affichée dans la culture actuelle dans un volet.
- **Progression** Un contrôle ProgressBar est affiché dans le contrôle **StatusStrip**.

Voir aussi

Référence

StatusStrip

Autres ressources

StatusStrip, contrôle

Validation de texte, exemple

Download sample

Cet exemple compare deux façons de valider du texte :

- L'utilisation d'expressions régulières dans un contrôle TextBox personnalisé
- L'utilisation d'un contrôle MaskedTextBox

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple montre comment vous pouvez créer un jeu de zones de texte spécialisées qui sont conçues pour un type spécifique d'entrée. Cet exemple inclut des zones de texte pour les adresses de messagerie, les adresses IP et les numéros de téléphone.

Le contrôle **MaskedTextBox** fournit la validation à mesure que l'utilisateur tape, de sorte qu'un texte incorrect n'apparaît jamais dans **MaskedTextBox**.

La classe `RegExTextBox`, dérivée de **TextBox** dans cet exemple, sert à la classe de base pour plusieurs autres contrôles. Cette classe valide le contenu du contrôle **TextBox** par rapport à une expression régulière donnée. Lorsque l'utilisateur clique sur le bouton **Valider** sur le formulaire principal, le code effectue une boucle sur la collection de contrôles pour rechercher tous les contrôles `RegExTextBox`, puis effectue un cast en ce type de base pour appeler la propriété `IsValid`. Si l'entrée n'est pas valide, la couleur du texte **TextBox** est rouge.

Voir aussi

Référence

Expressions régulières (Visual Studio)

Mise à jour de l'interface utilisateur à partir d'un thread, exemple

Download sample

Cet exemple montre comment mettre à jour l'interface utilisateur à partir d'un thread séparé.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

L'interface utilisateur ne peut pas être mise à jour à partir d'une procédure s'exécutant sur un thread qui n'est pas le thread d'interface utilisateur. Cet exemple montre comment mettre à jour l'interface utilisateur lorsque les données sont fournies depuis un thread différent.

Voir aussi

Autres ressources

Multithreading dans Visual Basic

Formulaires de niveau supérieur, exemple

Download sample

Cet exemple illustre une application semblable à Microsoft Word, composée de plusieurs formulaires de niveau supérieur.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5.

2. Créez plusieurs fenêtres de document à l'aide de la commande **Nouveau** dans le menu **Fichier** et vérifiez que tous les documents apparaissent dans la barre des tâches.

3. Utilisez le **Gestionnaire des tâches** pour vérifier qu'un seul processus est en cours d'exécution pour l'application.

Démonstrations

Beaucoup d'applications basées sur les documents prennent en charge plusieurs fenêtres, une par document ouvert, affichées dans la barre des tâches. La combinaison de touches ALT+TAB peut être utilisée pour basculer entre les documents, mais sans la charge mémoire provoquée en chargeant chaque document dans un processus séparé. Cet exemple implémente ce comportement.

La méthode `Sub Main`, située à l'intérieur d'une classe nommée `FormsManager`, constitue l'objet de démarrage de l'application. La méthode `Main` crée une nouvelle instance de formulaire en appelant la méthode `NewForm` de la classe `FormsManager`, puis ajoute le nouveau formulaire à un objet List. Une fois que le premier formulaire est créé, le code appelle System.Windows.Forms.Application.Run pour démarrer le principal thread de l'application afin que la fermeture du premier formulaire n'arrête pas le processus. Chaque formulaire de document expose un élément de menu **Nouveau**. Cliquer sur cet élément de menu appelle encore la méthode `NewForm` qui ouvre un nouveau formulaire de document tout comme `Sub Main` lors du chargement de l'application.

Chaque formulaire fournit également les éléments de menu **Fermer** et **Quitter**. Choisissez **Fermer** pour démarrer le processus de fermeture. Chaque formulaire a un événement Closing qui permet au code du formulaire de vérifier si le contenu du document a été modifié et, si tel est le cas, de demander à l'utilisateur s'il faut enregistrer ou non.

Si vous essayez de fermer un document modifié en fermant le formulaire ou en quittant l'application, une boîte de dialogue vous demandera d'enregistrer le contenu du formulaire. Si vous cliquez sur **Oui**, le code appelle la méthode `Save` du formulaire et ferme le formulaire. Si vous cliquez sur **Non**, le formulaire se ferme normalement. Si vous cliquez sur **Annuler**, le formulaire ne se ferme pas et un événement personnalisé est déclenché pour indiquer à l'application d'arrêter la fermeture. Cette application n'exécute pas réellement d'E/S de fichier pour enregistrer le fichier.

Voir aussi

Référence

System.Windows.Forms.Form.Closing

System.Windows.Forms.Application.Run

Héritage visuel, exemple

Download sample

Cet exemple montre comment vous pouvez utiliser l'héritage avec Windows Forms.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Sélectionnez **Générer** dans le menu **Générer** pour compiler le projet. Ainsi, vous pouvez consulter les formulaires dérivés dans le Concepteur Windows Forms.

2. Appuyez sur F5.

Démonstrations

Un formulaire est une classe comme toutes celles du .NET Framework, avec la capacité supplémentaire d'afficher des éléments d'interface utilisateur visuels. Pour utiliser ce type d'héritage, il vous suffit de créer une classe qui dérive d'une classe qui dérive à son tour de System.Windows.Forms.Form.

Trois formulaires dérivent de `BaseForm` : `MainForm`, `ControlSamples` et `MySamples`. La classe `BaseForm` inclut les contrôles Label et Button, ainsi que d'autres propriétés qui affectent l'apparence du formulaire. Les formulaires dérivés remplissent la zone centrale vide du formulaire soit avec des exemples d'utilisation de la nouvelle fonctionnalité **My**, soit avec des exemples des nouveaux contrôles Windows Forms. `MainForm` est l'objet de démarrage de l'exemple. Il démarre `ControlSamples` et `MySamples`.

Voir aussi

Tâches

Comment : ancrer des contrôles aux Windows Forms

Référence

Form

Autres ressources

Héritage visuel des Windows Forms

Navigateur Web, exemple

Download sample

Cet exemple crée un navigateur Web simple.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple est basé sur le contrôle WebBrowser. Des fonctionnalités supplémentaires sont ajoutées :

- **Navigation.** Un ToolStrip est utilisé pour les options **Précédent**, **Suivant**, **Actualiser**, **Accueil** et **Aller à**.
- **Sites communs.** Un **ToolStrip** avec des contrôles ToolStripButton est utilisé pour accéder à des sites spécifiques. Un deuxième formulaire est utilisé pour répertorier les sites indiqués. La liste de sites est conservée dans un fichier XML à l'aide de la classe XmlDocument.

Voir aussi

Référence

WebBrowser
ToolStrip
XmlDocument

Exemples de langages Visual Basic

Ces exemples illustrent la syntaxe et les fonctionnalités **My** Visual Basic.

Dans cette section

Appel API, exemple

Illustre l'appel aux fonctions de l'API Win32 pour énumérer et contrôler les processus et fenêtres, et interroger et modifier les paramètres système.

Tableau, exemple

Crée des tableaux de valeurs **Integer** et **Object**, et trie et recherche des tableaux.

Rappels, exemple

Illustre quatre méthodes d'implémentation des procédures de rappel.

Exceptions personnalisées, exemple

Crée des exceptions pour les erreurs spécifiques à une application.

DateTime et TimeSpan, exemple

Illustre le fonctionnement des méthodes DateTime et TimeSpan.

Accesseurs d'événement, exemple

Montre comment créer et utiliser un accesseur d'événement, lequel vous permet de personnaliser l'implémentation d'un événement.

Gestion des exceptions, exemple

Montre plusieurs options de gestion des exceptions dans les applications.

Système de fichiers, exemple

Montre l'objet My.Computer.FileSystem.

Génériques, exemple

Crée cinq types de collections génériques.

Programmation orientée objet, exemple

Illustre les options disponibles avec une programmation orientée objet.

Surcharge d'opérateur, exemple (Visual Basic)

Crée une classe qui représente l'argent, et implémente des opérateurs pour cette classe.

Mots clés overloads et overrides, exemple

Illustre Overloads et Overrides.

Mise en forme de chaînes, exemple

Permet aux utilisateurs de spécifier des options de mise en forme, puis affiche les résultats.

Méthodes String, exemple

Permet aux utilisateurs d'entrer des paramètres pour plusieurs méthodes String, puis affiche les résultats.

Fichiers temporaires, exemple

Crée et supprime des fichiers temporaires.

Fonctionnalités du langage Visual Basic 2005, exemple

Met en valeur de nouvelles fonctionnalités de langage Visual Basic.

Commentaires XML, exemple

Illustre la fonctionnalité de commentaires XML de Visual Basic.

Référence

Développement d'applications avec Visual Basic

Fournit des informations sur les structures de programme, l'accès aux données et l'éditeur Visual Basic.

Guide de programmation Visual Basic

Décrit les éléments du langage Visual Basic.

Rubriques connexes

Exemples d'applications Visual Basic

Ces exemples illustrent les tâches du projet, de l'utilisateur et de l'assembly.

Exemples de données

Ces exemples illustrent l'accès aux données.

Exemples de Windows Forms Visual Basic

Ces exemples présentent des applications Windows.

Exemples de composants serveur Visual Basic

Ces exemples illustrent des applications qui interagissent avec les composants du système d'exploitation Windows.

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Exemples d'applications Visual Basic

Cette page est le point de début pour tous les exemples Visual Basic inclus avec la documentation.

Appel API, exemple

Download sample

Cet exemple illustre les fonctions d'appel de l'API Win32 pour énumérer et contrôler les processus et fenêtres, et interroger et modifier les paramètres système.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

La classe `Win32API` est définie dans un fichier séparé, Win32API.vb. Chaque déclaration d'API, à l'aide de l'instruction Declare, est disponible en tant que membre **Shared** de la classe `Win32API`. Les membres partagés peuvent être appelés directement à l'aide du nom de classe. Il n'est pas nécessaire de créer une instance de la classe pour les utiliser.

Le formulaire principal contient un contrôle onglet avec quatre pages d'onglets. Chaque page d'onglets illustre une action à l'aide d'appels de l'API Win32 :

- **Processus actifs** Remplit une vue Liste avec tous les processus Windows en cours d'exécution L'accent est mis sur l'utilisation d'une fonction de rappel Win32, **EnumWindows**, à l'aide d'un délégué.

- **Fenêtres actives** Remplit un contrôle ListBox avec toutes les fenêtres actives, en filtrant les résultats de la fonction de rappel, **EnumWindows**, à l'aide de l'API Win32. Dans cet exemple, **EnumWindows** est défini à l'aide de la classe DllImportAttribute au lieu de l'instruction Declare pour marquer la différence entre l'utilisation des deux méthodes.

- **Afficher la fenêtre** Permet à l'utilisateur d'entrer une légende de fenêtre et/ou un nom de classe et de mettre la fenêtre au premier plan. Cet exemple accentue la surcharge de plusieurs types de paramètres vers une fonction API Win32 pour autoriser le passage de différents types de variables à la fonction.

- **Appels de l'API** Montre comment utiliser les appels de l'API Win32 suivants : **GetDiskFreeSpace**, **GetDiskFreeSpaceEx**, **GetDriveType**, **CreateDirectory**, **SwapMouseButton**, **IsPwrHibernateAllowed**, **SetSuspendState** et **Beep**. Ce sont des exemples d'appels qui retournent des informations qui ne peuvent pas être obtenues à partir des équivalents de la classe managée.

Voir aussi

Référence

Declare, instruction
DllImportAttribute

Tableau, exemple

Download sample

Cet exemple montre comment travailler avec des tableaux de types valeur et des tableaux de types référence. Le type valeur utilisé est le type **Integer**. Le type référence utilisé est une classe `Customer` définie dans le projet. La classe `Customer` expose deux membres publics, `Id` et `Name`.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Cliquez sur **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

La classe `Customer` implémente l'interface IComparable, laquelle est requise pour les fonctionnalités Sort et BinarySearch. Les objets `Customer` peuvent être triés sur la propriété `Name` ou sur la propriété `Id`. Le remplacement de **Chaînes** par **Objets** comme valeur de l'option **Tableau de** active le contrôle **Field to use for Sorts** qui détermine si un tableau d'objets `Customer` est trié sur le champ `Name` ou la propriété `Id` du client. La méthode `SetCompareKey` partagée de la classe `Customer` peut être appelée avant qu'un tri soit effectué afin de modifier le champ utilisé.

Les six boutons effectuent les tâches suivantes :

- **Create Static Array** Crée un tableau à l'aide de la syntaxe {} pour charger un tableau de valeurs. Une fois que le tableau est chargé, une procédure d'assistance, `DisplayArrayData`, énumère les valeurs du tableau et les met dans une liste. Pour plus d'informations sur la déclaration et l'initialisation des tableaux, consultez Dim, instruction (Visual Basic).

- **Sort** Charge les données en utilisant le même code que dans la procédure **Créer un tableau statique**. Une fois que les données sont chargées, la méthode partagée Sort de la classe Array est appliquée, puis les données sont chargées dans le contrôle `lstAfter` . Les chaînes et les types primitifs comme les entiers sont automatiquement comparables. L'interface **IComparable** est implémentée sur la classe `Customer` pour activer le tri.

- **Reverse** La procédure utilise la méthode Reverse pour inverser l'ordre des éléments dans le tableau. (**Reverse** ne produit pas un tri inversé des éléments ; il inverse l'affichage des éléments dans le tableau.)
- **Binary Search** L'exécution d'une recherche binaire nécessite que les éléments du tableau soient triés. Après que les données ont été chargées dans le tableau, elles sont triées, puis une recherche binaire est exécutée à l'aide de la méthode **BinarySearch**. La valeur fournie dans un contrôle zone de texte étiqueté **Binary Search** est utilisée comme critère de recherche. Si l'élément est trouvé, sa position d'index est alors affichée. Si l'élément n'est pas trouvé, la méthode **BinarySearch** retourne le complément de bits de l'endroit où l'élément se serait trouvé s'il avait existé.
- **Create Dynamic Array** Utilise la valeur numérique fournie dans le contrôle de zone de texte `txtLength` pour appliquer **ReDim** à un tableau. Il effectue ensuite une boucle sur chaque nouvel élément et affiche une zone de saisie demandant une valeur pour l'élément. Lorsque toutes les valeurs ont été obtenues, elles sont affichés.
- **Create Matrix Array** Génère un tableau à deux dimensions qui contient deux colonnes et trois lignes, puis l'affiche dans une zone de liste.

Voir aussi

Référence

Array Class
Array.Sort Method
Array.BinarySearch Method
Dim, instruction (Visual Basic)
IEnumerable Interface
Icomparable

Rappels, exemple

Download sample

Cet exemple permet à l'utilisateur d'effectuer un ensemble d'appels qui inclut un rappel vers le client pendant l'exécution. Il utilise des interfaces et des délégués.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le formulaire principal contient quatre boutons, chacun d'eux exécute une méthode de rappel à l'aide d'une technique de programmation différente.

- **Interface** Une interface, `ICallback`, avec une méthode, `CallbackMethod`, est définie. Le formulaire principal de l'exemple, `Form1`, implémente cette interface en définissant la méthode `CallbackMethod`. `Class1` contient une référence à une instance de `ICallback`. La référence est assignée par l'intermédiaire de sa méthode `RegisterInterFaceForCallback`. Lorsque la méthode `UseInterfaceCallback` de l'objet `Class1` est appelée, la méthode `Callback` de l'objet `ICallback` est exécutée. La référence à l'objet `ICallback` peut être supprimée en appelant la méthode `UnRegisterInterfaceForCallback`.

- **Délégué** Le projet contient une définition pour un délégué, `Delegate1`. `Class1` contient une référence, `del1`, à un objet `Delegate1`. À l'aide de la méthode `RegisterDelegateForCallback`, une instance d'un délégué peut être assignée à `del1`. Dans cet exemple, le délégué est la méthode `CallbackMethod` de `Form1`. Lorsque la méthode `UseDelegateCallback` de l'objet `Class1` est appelée, le `CallbackMethod` de `Form1` est exécuté. La méthode est exécutée à l'aide de la méthode `Invoke`. La référence au délégué peut être supprimée à l'aide de la méthode `UnRegisterDelegateForCallback` de `Class1`.

- **Asynchrone** La méthode `UseAsynchDelegateCallback` de `Class1` permet un appel asynchrone au délégué `del1`. L'appel est effectué à l'aide de la méthode `BeginInvoke` au lieu de la méthode `Invoke`.

- **Asynchrone vers le client** Une méthode de rappel est appelée sans assignation à l'objet de délégué, à l'aide d'un délégué AsyncCallback et de la méthode `BeginInvoke`.

Voir aussi

Référence

Delegate, instruction
Interface, instruction (Visual Basic)
Delegate
AsyncCallback

Exceptions personnalisées, exemple

Download sample

Cette solution illustre deux techniques plus avancées de gestion des exceptions : gestion des exceptions personnalisées et gestion globale des exceptions.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Dans l'**Explorateur de solutions**, cliquez avec le bouton droit sur le projet **Client**, puis sélectionnez **Définir comme projet de démarrage**.

2. Appuyez sur la combinaison de touches CTRL+F5. Le programme est le mieux exécuté à l'extérieur du débogueur pour consulter les effets d'un gestionnaire d'exceptions global.

Démonstrations

Cet exemple contient deux projets : un client Windows Forms et une bibliothèque de classes.

Bibliothèque de classes (serveur)

La bibliothèque de classes définit la classe `Customer` ainsi qu'un ensemble de classes d'exceptions personnalisées qui sont utilisées pour exposer des erreurs définies par l'application. Les classes d'exceptions sont liées par une hiérarchie d'héritage. La classe de base est `CRMSystemException`, laquelle hérite de ApplicationException. `CustomerException` et `EmployeeException` héritent de `CRMSystemException`. Il existe deux exceptions supplémentaires héritant de `CustomerException`:`CustomerNotFoundException` et `CustomerNotDeletedException`. Chaque classe expose différents niveaux d'informations et de fonctionnalités.

Code Client

L'application cliente fait référence à la bibliothèque de classes. Deux boutons de commande exécutent des méthodes sur la classe `Customer` et interceptent les exceptions personnalisées. Un troisième bouton de commande génère une exception non gérée. De plus, la case à cocher étiquetée "Turn on Global Exception Trap" active une interruption d'exception globale. Si vous activez cette case à cocher, le code défini dans la méthode OnThreadException sera exécuté à la place du gestionnaire Windows Form par défaut.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)

ApplicationException
ThreadException
OnThreadException

Autres ressources

Gestion structurée des exceptions en Visual Basic

DateTime et TimeSpan, exemple

Download sample

Cet exemple présente la structure DateTime et la structure TimeSpan.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5.

2. Tapez des valeurs dans les contrôles zone de texte appropriés, puis cliquez sur le bouton Actualiser de la page pour exécuter la méthode correspondante et afficher les résultats dans les contrôles label.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Sélectionnez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

Le formulaire contient un contrôle onglet avec plusieurs pages avec onglet. Chaque page avec onglet décrit un groupe de membres **DateTime** ou **TimeSpan**. Dans chaque onglet, les contrôles vous permettent de spécifier les paramètres d'entrée pour les différentes méthodes et propriétés.

Voir aussi

Référence

DateTime

TimeSpan

Accesseurs d'événement, exemple

Download sample

Cet exemple illustre la création et l'utilisation d'accesseurs d'événement. Cette fonctionnalité vous permet de personnaliser l'implémentation d'un événement et d'écrire une prise en charge **AddHandler**, **RemoveHandler** et **RaiseEvent** personnalisée. Cette fonctionnalité est souvent utilisée pour optimiser la mémoire utilisée par les contrôles serveur et pour les rendre plus robustes face à des appelants et des gestionnaires qui ne se comportent pas correctement.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple crée deux contrôles, `MemoryOptimizedBaseControl` et `ReliabilityOptimizedControl`. Chacun déclenche un événement `Click` toutes les cinq secondes. La différence entre les deux classes réside dans la façon dont les événements sont implémentés.

Contrôle personnalisé économe en mémoire Le contrôle `MemoryOptimizedBaseControl` utilise des accesseurs d'événement pour optimiser la mémoire consommée par la création d'événements sur le contrôle serveur. L'objet `EventList` interne est uniquement utilisé si le code appelant ajoute un gestionnaire d'événements pour l'événement `Click`. Une horloge interne est utilisée pour simuler des événements serveur déclenchés.

Contrôle personnalisé fiable `ReliabilityOptimizedControl` utilise des accesseurs d'événement pour créer un contrôle serveur plus fiable. Le contrôle serveur déclenche toujours l'événement `Click` de façon asynchrone à l'aide de `BeginInvoke`, ce qui garantit que les gestionnaires d'événements sur le client ne se bloquent pas les uns les autres. Cela signifie qu'un gestionnaire d'événements qui ne se comporte pas correctement n'empêche pas les autres d'être gérés. Une horloge interne est utilisée pour simuler des événements serveur déclenchés.

Voir aussi

Tâches

Comment : déclarer des événements qui économisent l'utilisation de la mémoire

Comment : déclarer des événements qui évitent les blocages

Référence

Event, instruction

Gestion des exceptions, exemple

Download sample

Cette solution illustre la gestion des exceptions structurée à l'aide de **Try...Catch...Finally**.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5 pour démarrer l'application en mode débogage à l'aide des points d'arrêt. Appuyez sur CTRL+F5 pour exécuter le programme et ignorer les points d'arrêt.

Démonstrations

Le code montre cinq variations à l'ouverture d'un fichier. Il y a cinq boutons de commande à tester. Chacun essaie d'ouvrir le fichier spécifié dans la zone de texte étiquetée **Text File To Open**. Chaque bouton, sauf celui qui est étiqueté **No Error Handling**, utilise plusieurs degrés de gestion des erreurs à l'aide des blocs **Try**, **Catch** et **Finally**.

- **No error handling** La classe FileStream est utilisée pour ouvrir le fichier spécifié dans le formulaire. Si le fichier n'existe pas, une exception est levée. En mode version finale, l'exécution du programme prend fin. En mode débogage, l'**Assistant Exception** s'affiche.

- **Basic error handling** L'appel destiné à ouvrir le fichier est encapsulé dans une instruction Try...Catch...Finally (Visual Basic) qui intercepte toutes les erreurs. Le message d'erreur s'affiche et l'exécution du programme continue.

- **Detailed error handling** De multiples clauses **Catch** sont utilisées pour fournir plus de détails à propos de l'erreur. En interceptant des erreurs spécifiques, le programme peut déterminer si le fichier n'a pas existé, si le dossier n'a pas existé ou si d'autres erreurs E/S se sont produites.

- **Message personnalisé** À l'aide de plusieurs clauses **Catch** et de la trace de la pile, le programme fournit un message d'erreur détaillé à propos de l'exception.

- **Try, Catch, Finally** La clause **Finally** est utilisée pour fermer le fichier s'il a été ouvert.

Voir aussi

Référence

Try...Catch...Finally, instruction (Visual Basic)
Exception

ApplicationException
StackTrace

Système de fichiers, exemple

Download sample

Cet exemple montre comment utiliser les méthodes de l'objet My.Computer.FileSystem.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Dans le menu **Générer**, choisissez **Générer FileSystemSample**.

2. Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Sélectionnez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

Cet exemple fournit une interface pour l'exécution des méthodes de système de fichiers de l'objet My.Computer.FileSystem. Un contrôle TreeView situé sur un côté vous permet de choisir le type de méthodes que vous souhaitez exécuter. Lorsqu'une méthode est sélectionnée, un contrôle personnalisé apparaît dans le formulaire qui vous permet d'entrer des paramètres de la méthode, puis exécute la méthode. Le projet contient donc un contrôle utilisateur séparé pour chaque méthode décrite.

Voir aussi

Référence

My.Computer.FileSystem, objet

Génériques, exemple

Download sample

Cet exemple montre comment créer plusieurs collections génériques et comment les manipuler.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

L'exemple illustre les concepts suivants :

- Utilisation des collections Queue, List, Stack, Dictionary et SortedList de l'espace de noms System.Collections.Generic.
- Méthode pour trier le contenu de la collection pour chaque type collection.
- Ajout et suppression d'éléments des structures de données.

L'exemple contient un formulaire. Au démarrage, le formulaire affiche deux ensembles de contrôles RadioButton, quatre contrôles Button et deux contrôles ListBox. Le contrôle **ListBox** de code source contient les données qui sont stockées. Les deux groupes **RadioButton** contrôlent le type de données et le type de collection. Les boutons exécutent les tâches effacer, charger, trier et inverser. La collection résultante est affichée dans le contrôle **ListBox** cible. Le code pour chaque type collection est organisé dans une directive #Region.

Voir aussi

Référence

System.Collections.Generic

Programmation orientée objet, exemple

Download sample

Cette solution illustre certaines des fonctionnalités orientées objet de Visual Basic.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Le projet comprend plusieurs définitions de classes. `Customer`, qui est la classe de base, contient uniquement trois propriétés. Quatre autres classes sont dérivées de la classe `Customer`. Chacune hérite des propriétés de `Customer`, mais ajoute une autre catégorie de membre de classe.

- `CustomerPropertySyntax` est une modification de la classe `Customer` qui définit la propriété `AccountNumber` pendant la construction. Une fois créé, `AccountNumber` ne peut pas être modifié.

- `CustomerWithConstructor` est une modification de `Customer` qui inclut un constructeur qui définit les trois propriétés pendant l'instanciation.

- `CustomerWithParameterizedProperty` est une personnalisation de `Customer` qui ajoute une propriété qui utilise un paramètre.

- `CustomerWithSharedMembers` ajoute une propriété partagée et une méthode partagée à la classe `Customer`.

Voir aussi

Référence

Class, instruction (Visual Basic)
Property, instruction
ReadOnly (Visual Basic)
Shared (Visual Basic)

Autres ressources

Présentation des classes

Surcharge d'opérateur, exemple (Visual Basic)

Download sample

Cette solution illustre les concepts et l'utilisation d'opérateurs surchargés en prenant Dollars américains et Euros pour exemple.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

L'exemple se compose de deux classes, `EURO` pour les euros et `USD` pour les dollars américains. `Form1` sert d'application de test pour les deux classes. Les utilisateurs peuvent entrer plusieurs valeurs pour `USD`, exécuter les opérateurs, puis convertir `USD` en `EURO`.

Les techniques de programmation illustrées sont les suivantes :

- **Opérateurs surchargés.** La classe `USD` possède les opérateurs surchargés suivants : constructeur, +, -, *, /.
- **Opérateurs Narrowing et Widening.** Les classes `EURO` et `USD` ont toutes deux une conversion étendue et restrictive vers et à partir de **Double**.

Voir aussi

Concepts

Conversions étendues et restrictives

Mots clés overloads et overrides, exemple

Download sample

Cet exemple montre comment étendre des classes dérivées avec des fonctionnalités de langage Visual Basic telles que la surcharge et la substitution. Il montre également comment autoriser plusieurs niveaux d'accès aux membres d'une classe, y compris **Public**, **Private** et **Protected**.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cette application simule un système d'enregistrement de véhicules, prenant en charge des automobiles, des bateaux et des bicyclettes.

L'application utilise une classe de base appelée `Vehicle` à partir de laquelle les classes `Automobile`, `Boat` et `Bicycle` sont dérivées. Chaque classe dérivée étend la classe de base à sa façon : en substituant des méthodes de la classe de base, en implémentant de nouvelles méthodes ou propriétés lui appartenant ou en remplaçant (en occultant) des membres de la classe de base. Il existe également une classe **Friend** appelée `DepartmentOfMotorVehicles` qui simule la lecture de données à partir d'une base de données et l'écriture de données sur cette dernière.

L'application illustre l'utilisation de ces instructions et modificateurs dans des classes et dans leurs membres :

Mot clé	Utilisation
Inherits, instruction	Utilisé sur `Automobile`, `Boat` et `Bicycle` pour indiquer que `Vehicle` est la classe de base.
NotInheritable	Utilisé sur la classe `DepartmentOfMotorVehicles`. Cette classe ne peut pas servir de classe de base.
MustInherit	Utilisé sur la classe `Vehicle`. Seules les classes qui dérivent de `Vehicle` peuvent être instanciées.
Overloads	La méthode `Register` dans la classe `Boat` est surchargée pour ajouter un nombre maximal de passagers.
Overridable	Utilisé dans la classe `Vehicle` sur la propriété `ID`. Alors qu'elles sont implémentées dans la classe de base, les classes dérivées peuvent implémenter une autre version.

Overrides	Utilisé dans les trois classes dérivées sur la propriété `CurrentValue`, pour remplacer l'implémentation par défaut dans la classe de base.
MustOverride	Utilisé dans les membres de classe `Vehicle` (`ComputeRegistrationFee` et `Salary`). Alors que ces membres ne sont pas implémentés dans `Vehicle`, ils doivent être implémentés dans les classes dérivées qui ne sont pas marquées `MustInherit`.
Shadows	Utilisé dans la classe `Bicycle` sur la méthode `Register`.
Public (Visual Basic)	Utilisé sur des classes et leurs membres pour indiquer qu'une classe ou un membre est disponible pour l'application cliente.
Protected (Visual Basic)	Utilisé sur les champs de la classe `Vehicle` . Ces champs sont accessibles dans la classe `Vehicle` et les classes dérivées, mais ne sont pas accessibles aux applications clientes.
Friend (Visual Basic)	Utilisé sur la classe `DepartmentOfMotorVehicles` . Cette classe est utilisée dans l'assembly, mais n'est pas accessible dans les applications clientes.
Private (Visual Basic)	Utilisé sur les champs de classe. Ces champs ne sont pas accessibles dans les applications clientes.
Shared (Visual Basic)	Utilisé sur les méthodes dans la classe `DepartmentOfMotorVehicles`. Cela permet à la classe `DepartmentOfMotorVehicles` d'agir en tant que bibliothèque de fonctions.

Voir aussi

Autres ressources

Programmation orientée objet dans Visual Basic

Mise en forme de chaînes, exemple

Download sample

Cet exemple d'application montre comment afficher des valeurs numériques, des valeurs date-heure et des valeurs d'énumération sous forme de chaînes à l'aide de divers codes de format standard et personnalisé spécifique à la culture.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Les options de mise en forme illustrées incluent :

- Des valeurs numériques mises au format de chaînes à l'aide de codes de format standard et personnalisé. Les modifications de mise en forme dépendent du CultureInfo sélectionné.

- Des valeurs de date/d'heure mises en forme en tant que chaînes à l'aide de codes de format standard et personnalisés. Les modifications de mise en forme dépendent du **CultureInfo** sélectionné.

- Des valeurs d'énumération mise au format de chaînes à l'aide de codes de format standard.

Voir aussi

Référence

CultureInfo Class
IFormatProvider Interface
IFormattable Interface
Object.ToString Method
Int32.ToString Method
DateTime.ToString Method
Enum.ToString Method

Concepts

Mise en forme pour différentes cultures

Méthodes String, exemple

Download sample

Cet exemple illustre plusieurs méthodes de manipulation de chaînes. Le but de l'exemple n'est pas de montrer comment appeler les fonctions, mais d'illustrer ce que les fonctions peuvent faire pour vous. Le formulaire principal contient un TabControl avec plusieurs pages d'onglets. Chaque page d'onglets est consacrée à une fonctionnalité de programmation String.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Pour afficher l'exemple de documentation

1. Dans l'**Explorateur de solutions**, double-cliquez sur le dossier **Documentation**.

2. Si vous utilisez Visual Basic Express, cliquez avec le bouton droit sur **ReadMe.htm** dans le dossier **Documentation**. Sélectionnez **Afficher dans le navigateur**.

3. Si vous utilisez une autre version de Visual Basic, double-cliquez sur **ReadMe.htm** dans le dossier **Documentation**.

Démonstrations

Le formulaire principal contient un **TabControl** avec les trois pages d'onglets qui décrivent des méthodes membres **String**, des méthodes partagées **String** et des méthodes StringWriter. Chaque page d'onglets permet à l'utilisateur d'entrer des valeurs de chaîne, puis d'exécuter des méthodes **String** en cliquant sur des boutons. Le design sous-jacent contient une classe `Method` et une classe `Parameter`. Chaque instance de la classe `Method` représente une méthode **String** différente. Ce design facilite le passage des valeurs entrées dans le formulaire à la méthode **String** appropriée.

Méthode	Description
System.String.Insert(System.Int32,System.String) System.String.Remove	Ces méthodes créent et retournent de nouveaux objets **String**. Beaucoup de ces méthodes sont surchargées et acceptent un, deux ou trois paramètres. Le code peut ignorer certains champs d'entrée dans le formulaire.

System.String.IndexOf System.String.StartsWith System.String.EndsWith	Ces méthodes retournent des informations sur une chaîne existante, mais ne créent ou ne modifient pas des objets **String**.
System.String.Format System.String.Join	Ces méthodes ont souvent besoin de deux **String** pour terminer une tâche ou créer de nouvelles chaînes et sont donc implémentées en tant que méthodes **Shared**.
System.Text.StringBuilder.ToString	La classe StringBuilder vous permet de manipuler les caractères de la chaîne. La méthode **ToString** récupère le texte contenu par l'objet **StringBuilder**.
System.IO.StringWriter.Write System.IO.TextWriter.WriteLine System.IO.StringWriter.ToString	La classe **StringWriter** est utile lorsque vous devez ajouter du texte à une chaîne de sortie. La classe **StringWriter** fournit une mémoire tampon interne dans laquelle vous pouvez écrire du texte comme si vous écriviez dans un fichier. Les méthodes **Write** et **WriteLine** ajoutent du texte à la mémoire tampon. La méthode **ToString** récupère le texte contenu par l'objet **StringWriter**.

Les boutons répertoriant les méthodes de classe **String** sont en fait des contrôles RadioButton. L'apparence des boutons est obtenue en attribuant la valeur Button à la propriété Appearance. Les contrôles ressemblent aux boutons, mais restent sélectionnés une fois que l'utilisateur a cliqué dessus.

Les boutons utilisés pour sélectionner les méthodes de la classe **String** appellent tous le même gestionnaire d'événements, `HandleCheckedChanged`. Cette procédure utilise de nombreuses clauses Handles. À l'intérieur de la procédure, une instruction **If...Then** utilise le paramètre d'expéditeur pour déterminer quel bouton a été sélectionné et agit de manière appropriée.

Il n'existe aucun moyen de faire flotter des contrôles au-dessus d'un contrôle onglet, afin qu'une seule instance d'un groupe de contrôles figure sur chaque page. Pour fournir cette fonctionnalité dans cet exemple, la sélection d'une page sur le contrôle onglet définit la page sélectionnée comme valeur de la propriété Parent d'un contrôle Panel contenant tous les contrôles « communs », comme suit :

```
pnlDemo.Parent = tabStringDemo.SelectedTab
```

Pour déclencher un point d'arrêt afin de pouvoir parcourir le code **StringBuilder** et **StringWriter**, l'exemple utilise la méthode System.Diagnostics.Debugger.Break. Cette méthode est appelée si le contrôle CheckBox étiqueté **Step through code** est sélectionné.

Voir aussi

Référence

Handles
String
StringBuilder
StringWriter
System.Diagnostics.Debugger.Break

Autres ressources

TabControl, contrôle (Windows Forms)

Fichiers temporaires, exemple

Download sample

Cet exemple montre comment créer et utiliser des fichiers temporaires, et comment localiser le répertoire \Temp sur l'ordinateur de l'utilisateur.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Ce didacticiel illustre les tâches suivantes :

- Recherche du répertoire \Temp sur l'ordinateur d'un utilisateur.
- Création d'un fichier temporaire à l'aide de la méthode System.IO.Path.GetTempFileName.
- Affectation de Temporary à l'attribut d'un fichier temporaire afin d'améliorer les performances.
- Suppression d'un fichier temporaire.

Voir aussi

Référence

Path.GetTempFileName Method
Path Class
StreamReader Class
StreamWriter Class
System.IO Namespace

Fonctionnalités du langage Visual Basic 2005, exemple

Download sample

Cet exemple montre les nouvelles fonctionnalités de langage de Visual Basic, comprenant les génériques, la surcharge d'opérateur et le mot clé **Using**.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple montre ces fonctionnalités de langage :

- **Surcharge d'opérateur** La classe `ValidatedString` représente une chaîne avec validation personnalisée. L'opérateur `&` retourne la concaténation de deux objets `ValidatedString`.

- **Génériques** La classe `Pair` contient deux valeurs. Lorsque vous déclarez une instance de la classe `Pair`, vous spécifiez le type des deux valeurs. La méthode `Match` teste si les types et les valeurs de la paire sont identiques.

- **Using** L'instruction Using (Visual Basic) fait référence à une ressource de flux de données ouverte sur une demande d'un client Web. L'instance de flux est automatiquement supprimée à la fin du bloc **Using**.

- **TryCast et IsNot** TryCast essaie d'effectuer un cast d'un objet à un type particulier et retourne la valeur convertie. Si le cast échoue, **Nothing** est retourné. L'opérateur **IsNot** est utilisé pour tester le résultat de retour pour **Nothing**.

Voir aussi

Référence

Operator, instruction

Using, instruction (Visual Basic)

Concepts

Types génériques en Visual Basic

Autres ressources

Exemples de langages Visual Basic

Commentaires XML, exemple

Download sample

Cet exemple montre la fonctionnalité de commentaire XML de Visual Basic.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Cet exemple montre comment créer des commentaires XML, les consulter dans l'éditeur et y accéder par programme.

La classe `Automobile` définit les éléments de type suivants, chacun d'entre eux montrant un aspect différent des commentaires XML :

- **Types** `Automobile` et `EngineOption`
- **Propriétés** `Model`
- **Méthodes** Constructeurs, `Upgrade`, `CalculateMaxTowing`, `WillItTow`
- **Événements** `NeedsTuneUp`
- **Délégués** `RotateTires`, `ReplaceSparkPlugs`
- **Champs** `car`, `modelValue`

Le gestionnaire d'événements `Click` du bouton **Create Car** appelle plusieurs des membres `Automobile`. Lors de l'utilisation de l'IntelliSense dans l'**éditeur de code**, les commentaires XML sont affichés comme info-bulles.

Le gestionnaire d'événements `Click` du bouton **List Summaries** utilise la classe XmlDocument pour interroger le fichier XML qui contient les commentaires XML. Lorsque le projet est compilé, ce fichier XML est créé et stocké avec l'assembly du projet. L'onglet **Compiler** du **Concepteur de projets** vous permet de spécifier l'emplacement et le nom de ce fichier.

Voir aussi

Tâches

Comment : créer une documentation XML en Visual Basic

Exemples de composants serveur Visual Basic

Ces exemples illustrent des tâches courantes à l'aide de composants.

Dans cette section

Port COM, exemple

Montre comment contrôler un port de communication.

Journal des événements, exemple

Montre comment créer et supprimer des journaux d'événements, et comment lire et écrire des données dans des journaux d'événements système et des journaux d'événements personnalisés.

Notification de fichier, exemple

Utilise la classe FileSystemWatcher pour répondre quand les fichiers sont créés, supprimés, modifiés ou renommés.

Gestion des processus, exemple

Utilise des objets dans l'espace de noms System.Diagnostics pour rassembler des informations sur un processus en cours.

File d'attente des messages, exemple

Montre comment envoyer et recevoir des messages de file d'attente de messages d'applications Windows Forms.

Compteurs de performance, exemple

Montre comment lire et écrire dans les compteurs de performance.

Contrôle des processus, exemple

Illustre certaines manières de lancer et de surveiller des processus à l'aide de la classe Process de l'espace de noms **System.Diagnostics**.

Réponse à des événements système, exemple

Utilise la classe SystemEvents, qui permet à une application de réagir aux événements déclenchés par l'activité du système.

Gestionnaire de services, exemple

Montre comment contrôler les services Windows.

WMI, exemple

Utilise des objets dans l'espace de noms System.Management pour accéder à Windows Management Instrumentation (WMI).

Référence

Process

Cette classe est utilisée pour démarrer et arrêter des processus.

FileSystemWatcher

Cette classe déclenche des événements lorsque des fichiers sont modifiés.

SystemEvents

Cette classe déclenche des événements lorsque les paramètres d'affichage sont modifiés.

Rubriques connexes

Exemples d'applications Visual Basic

Ces exemples illustrent les tâches du projet, de l'utilisateur et de l'assembly.

Exemples de données

Ces exemples illustrent l'accès aux données.

Exemples de Windows Forms Visual Basic

Ces exemples illustrent des applications Windows Forms.

Exemples de langages Visual Basic

Ces exemples illustrent des concepts du langage Visual Basic.

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Exemples d'applications Visual Basic

Cette page est le point de début pour tous les exemples Visual Basic inclus avec la documentation.

Services d'annuaire, exemples

Contient des exemples qui accèdent à Active Directory, et le gèrent, montrant comment effectuer plusieurs opérations d'annuaire typiques.

Composants serveur pour applications .NET Framework

Examine les composants serveur fournis par le .NET Framework, tel que MessageQueue et EventLog.

Port COM, exemple

Download sample

Cet exemple montre comment contrôler un port de communication.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Configuration requise

Cette application requiert un modem installé sur l'un des ports COM.

Démonstrations

- Comment déterminer les ports COM disponibles.
- Comment utiliser un port COM pour communiquer avec un modem.
- Comment utiliser des appels de l'API Win32 pour contrôler la communication avec le port COM.

Voir aussi

Tâches

Comment : passer des appels avec des modems attachés aux ports série dans Visual Basic

Autres ressources

Accès aux ports de l'ordinateur

Journal des événements, exemple

Download sample

Cet exemple illustre la création et la suppression de journaux des événements ainsi que la lecture et l'écriture dans des journaux des événements système et des journaux des événements personnalisés.

Visual Basic propose deux mécanismes d'enregistrement différents :

- Le composant EventLog, utilisé dans cet exemple, fournit l'accès aux journaux des événements du système d'exploitation. Vous pouvez ajouter et supprimer des journaux, ajouter et supprimer des sources de journaux d'événements, écrire et supprimer des messages.
- L'objet **My.Application.Log** fournit également l'accès à certaines des fonctionnalités **EventLog**. De plus, **My.Application.Log** possède des fonctionnalités qui vous permettent d'écrire des messages dans des fichiers texte. Pour plus d'informations, consultez Enregistrement dans un journal, exemple et Enregistrement d'informations provenant de l'application.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

  Tous les systèmes d'exploitation ne prennent pas en charge les journaux des événements. Pour plus d'informations, consultez **EventLog**.

Démonstrations

Le formulaire principal, `Form1`, contient trois contrôles Button pour lire, écrire et créer ou supprimer des journaux des événements. Chacun ouvre un nouveau formulaire qui collecte des informations spécifiques à l'opération.

Le formulaire `WriteForm` collecte le texte, l'ID et le type pour une entrée du journal des événements et écrit l'entrée dans le journal des événements de l'application à l'aide de la méthode WriteEntry.

Le formulaire `ReadForm` remplit un contrôle ListBox avec les noms des journaux des événements sur l'ordinateur. La liste est obtenue en appelant la méthode GetEventLogs, puis en affichant la valeur de la propriété LogDisplayName pour chaque journal. Les dix dernières entrées du journal sélectionné sont récupérées à l'aide de la propriété Entries, puis affichées dans un contrôle RichTextBox.

Le formulaire `CreateDeleteForm` appelle les méthodes CreateEventSource et Delete. La méthode SourceExists est utilisée pour

vérifier que le journal des événements et la source n'existent pas déjà avant de les créer. La méthode Exists est utilisée pour vérifier que le journal des événements existe avant de le supprimer.

Voir aussi

Référence

EventLog

System.Diagnostics

Concepts

Introduction au composant EventLog

Enregistrement d'informations provenant de l'application

Autres ressources

Enregistrement dans un journal, exemple

Notification de fichier, exemple

Download sample

Cet exemple utilise la classe FileSystemWatcher pour émettre une réponse une fois les fichiers créés, supprimés, modifiés ou renommés.

L'exemple de formulaire vous permet de définir des propriétés de la classe **FileSystemWatcher**, notamment :

- Path Chemin d'accès de système de fichiers à "suivre" par l'objet.
- Filter Types de fichiers à suivre. La valeur par défaut *.*.
- NotifyFilter Types de modifications à suivre. Plusieurs valeurs, ou aucune, provenant de l'énumération System.IO.NotifyFilters peuvent éventuellement être combinées pour être affectées à cette propriété. Par défaut, **FileSystemWatcher** détecte les modifications de FileName, DirectoryName et de LastWrite.
- IncludeSubdirectories Indique si la surveillance doit inclure des répertoires sous le chemin d'accès sélectionné.
- EnableRaisingEvents Si la valeur définie est **true**, l'objet tente de déceler les modifications apportées au système de fichiers et déclenche des événements lorsqu'elles ont lieu. Fondamentalement, il s'agit du commutateur marche/arrêt. Dans cette démonstration, cette propriété est contrôlée à l'aide d'un contrôle CheckBox affiché comme un bouton bascule.

Security Note Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5.
2. Utilisez les contrôles de droite pour modifier les fichiers suivis.
3. Utilisez l'**Explorateur Windows** pour modifier les fichiers suivis. La technique de modification des fichiers n'a aucune influence.

Démonstrations

Les contrôles sur le formulaire vous permettent de générer des événements en créant, supprimant, renommant et modifiant des fichiers. Cette barre d'état des composants du formulaire inclut un composant **FileSystemWatcher**, localisé sous l'onglet **Composants** de la **boîte à outils**. L'utilisation du composant rend plus facile la définition des propriétés au moment du design dans la fenêtre **Propriétés**.

Les boutons **Create**, **Rename**, **Modify** et **Delete Sample File** situés sur le formulaire permettent de générer les événements **FileSystemWatcher**. Ces boutons créent des fichiers temporaires dans le dossier sélectionné. Utilisez le bouton **Delete All**

Sample Files pour nettoyer tous les fichiers d'exemple restants lorsque vous avez terminé. Cinq des événements **FileSystemWatcher** sont gérés dans cet exemple :

- Created, Changed et Deleted Ces trois événements sont tous gérés par le même gestionnaire d'événements,`HandleChangedCreatedDeleted`. Les détails de l'événement sont affichés sur le formulaire.
- Renamed Les noms anciens et nouveaux sont affichés.
- Error Le message d'exception est affiché.

Voir aussi

Référence

FileSystemWatcher
NotifyFilters

Gestion des processus, exemple

Download sample

Cet exemple utilise des objets dans l'espace de noms System.Diagnostics pour rassembler des informations sur un processus en cours.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5.

2. Cliquez sur un processus pour charger ses détails et données de thread. Utilisez les menus pour afficher la liste de modules chargée par un processus particulier. Appuyez sur F5 au cours de l'exécution du programme pour actualiser la liste de processus.

Démonstrations

Les processus sur l'ordinateur actuel sont affichés dans un contrôle ListView en appelant la méthode GetProcesses. Lorsqu'un processus est sélectionné, les détails sur ce processus s'affichent dans un autre contrôle **ListView**. Les détails de processus sont récupérés à l'aide de l'instance de Process qui représente le processus. La propriété Threads est utilisée pour énumérer les threads du processus qui sont ensuite affichés dans un autre contrôle **ListView**. Chaque thread est représenté par une instance de ProcessThread. Les modules pour chaque processus peuvent être inspectés en sélectionnant un processus, puis en cliquant sur **Modules** dans le menu principal ou le menu contextuel. Un formulaire séparé affiche les modules chargés par le processus et inclut le nom du fichier où le code a été chargé.

Trois processus ne fournissent pas d'informations complètes :

- `_Total` Ce processus est ajouté dans le code et ne représente pas de processus. Il est utilisé pour fournir un résumé du temps d'exécution et de l'utilisation du processeur.

- `Idle` Bien qu'il soit retourné comme un processus, il ne s'agit pas d'un processus capable de retourner des données.

- `System` Ce processus n'expose pas d'informations de module.

Il peut arriver que vous n'ayez pas accès à l'ensemble des informations d'un processus. Dans ce cas, un message s'affiche.

Voir aussi

Référence

Process

System.Diagnostics
ProcessThread
Threads

File d'attente des messages, exemple

Download sample

Cet exemple montre comment envoyer et recevoir des messages de file d'attente de messages d'applications Windows Forms.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'affiche.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier solution MSMQListener.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Appuyez sur F5 pour compiler et exécuter le projet MSMQListener.
2. Ouvrez une autre instance de Visual Studio et ouvrez le fichier solution MSMQClient.
3. Appuyez sur F5 pour compiler et exécuter le projet MSMQListener.

Démonstrations

L'application envoie des messages à une file d'attente des messages privée. Les messages sont reçus un par un et traités par le MSMQListener. Cet exemple a deux composants fonctionnant ensemble :

- Un client Windows Forms qui envoie des messages. Il s'agit d'une application simple qui laisse l'utilisateur publier des commandes dans une file d'attente des messages privée.
- Un écouteur Windows Forms qui traite les messages envoyés par le client. L'écouteur reçoit d'abord tous les ordres qui sont actuellement dans la file d'attente. Puis, il écoute les ordres supplémentaires et les traite au fur et à mesure qu'ils entrent. L'écouteur répond à un événement ReceiveCompleted lorsqu'un message a été reçu.

Voir aussi

Tâches

Comment : créer des instances du composant MessageQueue

Référence

MessageQueue

Compteurs de performance, exemple

Download sample

Cet exemple montre comment lire et écrire dans des compteurs de performance. Ces compteurs peuvent être utilisés pour suivre la performance du système d'exploitation et des applications .NET Framework. Des compteurs système et des compteurs personnalisés sont utilisés.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

1. Créez un compteur de performance personnalisé dans lequel il est possible d'écrire.

   a. Ouvrez l'**Explorateur de serveurs**.

   b. Ouvrez le nœud **Serveurs**, puis le nœud de serveur pour votre ordinateur, ouvrez ensuite le nœud **Compteurs de performance**.

   c. Cliquez avec le bouton droit sur le nœud **Compteurs de performance**, puis cliquez sur **Créer une nouvelle catégorie**.

   d. Dans la boîte de dialogue Générateur de compteurs de performance, entrez **exemple de compteur de performance** sous **Nom de la catégorie** et une description dans la zone de texte **Description de la catégorie**.

   e. Cliquez sur le bouton **Nouveau** pour ajouter un nouveau compteur de performance.

   f. Tapez un nom et une description pour le compteur. Choisissez **NumberOfItems32** pour le **Type**.

2. Appuyez sur F5 pour exécuter le projet.

3. Choisissez le compteur de performance personnalisé que vous avez généré précédemment. Choisissez-le par **Catégorie** et **Compteur**, puis utilisez les boutons pour incrémenter et décrémenter la valeur.

Démonstrations

Les tâches décrites comprennent :

- **Liste des catégories** Les catégories de compteurs de performance sont récupérées à l'aide de la méthode GetCategories et affichées dans un contrôle ComboBox.

- **Liste des compteurs** Les compteurs de performance sont récupérés à l'aide de la méthode GetCounters et affichés dans un contrôle **ComboBox**. Seuls les compteurs de la catégorie sélectionnée sont récupérés.

- **Extraction de données** La méthode NextValue est utilisée pour récupérer la valeur actuelle du compteur sélectionné.
- **Compteurs Personnalisés** L'application vous permet seulement d'incrémenter des compteurs personnalisés. Un compteur personnalisé est défini comme un compteur dans lequel vous ne pouvez pas appeler la méthode **NextValue**.

Voir aussi

Référence

PerformanceCounter

System.Diagnostics

Concepts

Introduction à l'analyse des seuils de performance

Introduction à l'analyse des seuils de performance

Contrôle des processus, exemple

Download sample

Cet exemple illustre certaines façons de lancer et de surveiller des processus à l'aide de la classe Process de l'espace de noms System.Diagnostics.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

La classe **Process** est utilisée pour :

- exécuter une application avec et sans arguments de ligne de commande, à l'aide de la méthode Start ;
- répertorier toutes les applications en cours d'exécution, à l'aide de la méthode GetProcesses ;
- répertorier les modules chargés, à l'aide de la propriété Modules ;
- récupérer des informations relatives à l'application en cours d'exécution, à l'aide de la méthode GetCurrentProcess.

L'exemple décrit également certaines techniques de programmation avec StringBuilder et les classes List génériques.

Voir aussi

Référence

Process
System.Diagnostics
Start
GetProcesses
Modules
GetCurrentProcess
StringBuilder

Réponse à des événements système, exemple

Download sample

Cet exemple utilise la classe SystemEvents qui permet à une application de réagir aux événements déclenchés par l'activité du système.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

La classe **Microsoft.Win32.SystemEvents** fournit plusieurs événements publics que vous pouvez recevoir dans votre code. Cet exemple montre cinq des événements disponibles de la classe **SystemEvents**. Activez la case à cocher du formulaire pour ajouter un gestionnaire d'événements pour l'événement. Le tableau suivant fournit des explications sur les événements et la manière de les déclencher.

Modification	**Événement**	**Comment déclencher un événement**
Écran	DisplaySettingsChanged	Modifiez la résolution de l'écran.
Heure	TimeChanged	Modifiez l'heure système actuelle.
Mode d'alimentation	PowerModeChanged	Lors de l'exécution sur un ordinateur portable, modifiez l'état connecté.
Préférence de l'utilisateur	UserPreferenceChanged	Modifiez un paramètre Windows. Par exemple, cliquez avec le bouton droit sur le bureau pour afficher la boîte de dialogue **Propriétés d'affichage** et modifier un paramètre.
Polices installées	InstalledFontsChanged	Installez ou supprimez une police.

Voir aussi

Référence

SystemEvents

DisplaySettingsChanged
TimeChanged
PowerModeChanged
UserPreferenceChanged
InstalledFontsChanged

Gestionnaire de services, exemple

Download sample

Cet exemple montre comment contrôler des services Windows.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Les services Windows sont des programmes à durée d'exécution longue qui effectuent leurs tâches sans intervention de l'utilisateur. Cet exemple utilise la classe ServiceController de l'espace de noms System.ServiceProcess. Cette classe vous permet de récupérer une liste des services en cours d'exécution via une méthode partagée, GetServices. À l'aide de la classe **ServiceController**, vous pouvez démarrer, arrêter, suspendre ou reprendre un service.

À l'aide de l'interface utilisateur présentée, vous pouvez démarrer, arrêt, suspendre et reprendre un service. L'application vérifie l'état actuel du service pour voir quels boutons doivent être activés. La commande **Relist All Services** du menu **Actions** appelle la méthode **GetServices** pour remplir le contrôle ListView avec les processus. La commande **Refresh** du menu **Actions** met à jour l'état de chaque processus déjà répertorié dans le contrôle **ListView**.

Voir aussi

Référence

ServiceController

System.ServiceProcess

Autres ressources

Surveillance des services Windows

WMI, exemple

Download sample

Cet exemple utilise des objets dans l'espace de noms System.Management pour accéder à Windows Management Instrumentation (WMI).

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

Les deux pages avec onglet de TabControl présentent deux types de tâches WMI :

- Les **requêtes** Ce type montre comment utiliser l'objet ManagementObjectSearcher pour effectuer une requête sur les informations système suivantes : nom du système d'exploitation, version, fabricant, nom d'ordinateur, répertoire Windows, fabricant du système informatique, modèle, type du système et mémoire physique totale, nom du processeur; version du BIOS et fuseau horaire. Ces requêtes sont créées à l'aide de la classe SelectQuery.

- Les **objets** Ce type montre comment effectuer une énumération parmi les classes WMI de niveau supérieur ainsi que tous leurs objets de gestion de données à l'aide des classes ManagementClass et EnumerationOptions.

Voir aussi

Référence

ManagementObject
ManagementObjectSearcher
SelectQuery
ManagementOperationObserver
ManagementClass
EnumerationOptions

Exemples de sécurité Visual Basic

Ces exemples illustrent des tâches de sécurité.

Dans cette section

Algorithmes de hachage de chiffrement, exemple

Utilise des algorithmes de hachage.

Algorithmes symétriques, exemple

Utilise des algorithmes symétriques pour le chiffrement.

Référence

HashAlgorithm

Cette classe est utilisée pour calculer des algorithmes de hachage.

SymmetricAlgorithm

Cette classe est utilisée pour chiffrer des données.

Rubriques connexes

Déploiement ClickOnce OnDemand, exemple de technologie

Montre le déploiement d'applications ClickOnce à l'aide de classes dans l'espace de noms System.Deployment.Application.

Mise à jour d'arrière-plan ClickOnce, exemple de technologie

Montre l'utilisation de l'API ClickOnce pour télécharger une mise à jour d'application ClickOnce en tâche en arrière-plan.

Chiffrement à clé publique, exemple de technologie

Montre comment vous pouvez utiliser le chiffrement à clé publique pour échanger des messages.

Services de chiffrement

Présente les concepts clés du chiffrement et décrit son implémentation dans le .NET Framework.

Indications de codage sécurisé

Discute comment utiliser le .NET Framework pour dresser des barrières de protection contre les codes nuisibles et les actions indésirables provenant de l'extérieur.

Exemples d'applications Visual Basic

Ces exemples illustrent les tâches du projet, de l'utilisateur et de l'assembly.

Exemples de données

Ces exemples illustrent l'accès aux données.

Exemples de Windows Forms Visual Basic

Ces exemples illustrent des applications Windows Forms.

Exemples de langages Visual Basic

Ces exemples illustrent des concepts du langage Visual Basic.

Exemples de composants serveur Visual Basic

Ces exemples illustrent des composants serveur.

Exemples d'applications Visual Basic

Cette page est le point de début pour tous les exemples Visual Basic inclus avec la documentation.

Algorithmes de hachage de chiffrement, exemple

Download sample

Cet exemple illustre l'utilisation des fonctions de hachage pour garantir l'intégrité des données.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour ouvrir le fichier d'exemple dans l'Explorateur de solutions

1. Cliquez sur **Télécharger l'exemple**.

   Le message **Téléchargement de fichier** s'affiche.

2. Cliquez sur **Ouvrir**, puis dans la colonne de gauche de la fenêtre du dossier du zip, cliquez sur **Extraire tous les fichiers**.

   L'**Assistant Extraction** s'ouvre.

3. Cliquez sur **Suivant**. Vous pouvez modifier le répertoire dans lequel les fichiers seront extraits, puis cliquer de nouveau sur **Suivant**.

4. Assurez-vous que la case à cocher **Afficher les fichiers extraits** est activée, et cliquez sur **Terminer**.

5. Double-cliquez sur le fichier .sln de l'exemple.

   La solution exemple est affichée dans l'**Explorateur de solutions**. Vous pouvez obtenir un avertissement de sécurité indiquant que l'emplacement de la solution n'est pas fiable. Cliquez sur **OK** pour continuer.

Pour exécuter cet exemple

- Appuyez sur F5.

Démonstrations

L'application compare deux valeurs de hachage : la valeur de hachage du texte dans un fichier avec la valeur de hachage du texte dans un contrôle TextBox. Les valeurs de hachage sont des représentations numériques uniques et compactes d'une portion de données. Le fichier texte est une version XML de la table `Products` de la base de données Northwind. Lorsque l'application démarre, le fichier XML, stocké dans **My.Resources**, est récupéré et chargé dans le contrôle **TextBox**. La valeur de hachage pour le fichier XML est calculée. Lorsque vous cliquez sur le bouton **Comparer !**, la valeur de hachage du texte **TextBox** est calculée. Si vous n'avez pas modifié le texte dans le contrôle **TextBox**, les valeurs de hachage correspondront. Même la modification d'un seul caractère dans un fichier source modifie la valeur de hachage.

Les contrôles RadioButton déterminent le type de hachage calculé : MD5, SHA1 ou SHA384. Une classe .NET Framework séparée, dérivée de la classe HashAlgorithm, existe pour chaque algorithme de hachage. Ces classes sont utilisées pour calculer les valeurs de hachage.

Voir aussi

Référence

MD5CryptoServiceProvider
SHA1CryptoServiceProvider
SHA384Managed
StreamReader
WriteXml
ReadXml

Concepts

Vue d'ensemble du chiffrement

Algorithmes symétriques, exemple

Download sample

Cet exemple utilise les classes Rijndael et TripleDESCryptoServiceProvider pour implémenter un chiffrement symétrique (à clé secrète), comme décrit dans la Vue d'ensemble du chiffrement.

Remarque de sécurité
Cet exemple de code est fourni pour illustrer un concept et ne doit pas être utilisé dans des applications ou des sites Web, car il peut ne pas illustrer les pratiques de programmation les plus sûres. Microsoft n'assume aucune responsabilité pour tout dommage indirect ou consécutif en cas d'utilisation de l'exemple de code à des fins autres que celles prévues.

Pour exécuter cet exemple

1. Cliquez sur le bouton **Télécharger l'exemple** dans cette page. Dans la boîte de dialogue **Téléchargement de fichier**, cliquez sur **Ouvrir**.
2. Enregistrez les fichiers sur votre ordinateur.
3. Dans le menu **Fichier** de Visual Studio, pointez sur **Ouvrir**, puis cliquez sur **Projet/Solution**.
4. Recherchez le dossier où vous avez enregistré les fichiers et sélectionnez le fichier solution. Cela ouvre la solution dans Visual Studio.
5. Appuyez sur F5.

Démonstrations

La classe `SampleCrypto` encapsule l'algorithme de chiffrement, le salt et le vecteur d'initialisation (IV) ainsi que les routines de chiffrement et de déchiffrement qui définissent un fichier chiffré. Le constructeur accepte un paramètre qui détermine le type de chiffrement, Rijndael ou TripleDES. Le champ correspondant dans la classe, `crpSym`, est du type abstrait SymmetricAlgorithm. La valeur qui lui est affectée est une instance de la classe **Rijndael** ou de la classe **TripleDESCryptoServiceProvider**. La méthode `CreateSaltIVFile` enregistre les valeurs salt et IV dans un fichier .dat. Les méthodes `Decrypt` et `Encrypt` traitent le fichier source qui est spécifié dans la propriété `SourceFileName`.

Une instance de la classe `SampleCrypto` est créée lorsque le formulaire est chargé. Les contrôles du formulaire illustrent les méthodes et les propriétés de la classe `SampleCrypto`. Lorsque l'option **Encrypt with Password** est activée, la clé n'est ni générée, ni définie automatiquement, mais dérivée d'un mot de passe qui a fait l'objet d'un salt. De plus, le salt et le vecteur d'initialisation (IV) persistent non chiffrés dans un fichier .dat. De cette façon, vous pouvez voir comment vous pouvez envoyer un document à quelqu'un en toute sécurité : le document chiffré peut être envoyé via une connexion non sécurisée, et le fichier .dat et le mot de passe peuvent être transférés en toute sécurité à l'aide de l'algorithme de chiffrement asymétrique (ou à clé publique). Ce type de chiffrement est la forme de chiffrement la plus sécurisée, mais demande un temps de traitement bien plus long. C'est pourquoi il est généralement utilisé pour les petits éléments tels qu'une clé secrète, un fichier salt/IV ou un mot de passe.

Voir aussi

Référence

TripleDESCryptoServiceProvider
Rijndael
SymmetricAlgorithm
Salt
IV

Concepts

Modèle de chiffrement de .NET Framework